MONASH University

Efficient Spatial Query Processing in Urban Environments

M G Gaminda Chaluka Ruwanga Salgado

A thesis submitted for the degree of Doctor of Philosophy at

Monash University in 2019

Clayton School of Information Technology

Copyright Notice

Abstract

Recent advances in telecommunication technologies and the increasing availability of location sensitive mobile devices have substantially enhanced the user experience in location-based services (LBSs). LBSs are the services that take into account the geographic locations (i.e., spatial coordinates) of users to provide tailored information. In this thesis, we study a variety of location-based queries in urban environments. Mainly, we focus on processing such queries in the indoor space motivated by the abundance of applications of indoor location-based services and lack of existing studies supporting indoor spatial queries. Although the spatial query processing has been extensively studied for outdoor space, these techniques are not efficient when applied to indoor venues due to inherent differences in the topology of indoor space [1]. Next, we briefly describe the contributions we make in this thesis.

We study a variant of route planning query called category aware multi-criteria route planning queries (denoted by CAM) that take into account not only route length, but also other relevant attributes such as total price and total waiting time etc., in determining an optimal route. Although the problem of CAM query is NP-hard in the number of query categories, we propose exact solutions suitable when the number of query categories is small which is typically the case in real-world applications. To handle the case when the number of query categories is large, we provide a fast approximation solution based on a novel *dominance-based* pruning technique.

We are the first to study skyline routes search in the indoor space. We take into account two attributes, route distance and the number of shops/stores

visited, to determine the dominance of a route over another. We prove that the problem is NP-hard in the number of query keywords. We propose an exact solution, assuming the number of query keywords is small. The experiments on a real-world dataset show the efficiency and scalabilty of the proposed solution.

Furthermore, we are also the first to investigate continuous detour queries in the indoor space. We propose techniques that exploit the geometric properties of the hyperbolas and the unique properties of the indoor space such as rooms, hallways, etc. The key idea behind our solution is to utilize the safe zones to reduce the number of re-evaluation of the detour queries against users' movements. The experiments show that our solution can process a continuous detour query significantly faster than a competitor and also significantly reduces the communication overhead.

We introduce a variant of spatial keyword query called continuous range spatial keyword queries over moving spatio-textual objects (or $CRSK$-mo queries) that returns objects within a given range where the range is determined based on both spatial proximity and textual similarity. We exploit the spatial and textual upper bounds between queries and objects to form safe zones (at the client-side) and buffer regions (at the server-side) to reduce both communication and computation overhead. The experimental results demonstrate the superior performance of the proposed solution.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature:

Print Name: M G Gaminda Chaluka Ruwanga Salgado

Date: July 29, 2019

Publications during Enrollment

1. Salgado, C., Cheema, M. A., & Taniar, D. (2018, November). "An efficient approximation algorithm for multi-criteria indoor route planning queries". In Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (pp. 448-451). ACM.

2. Salgado, Chaluka, Muhammad Aamir Cheema & David Taniar. "Category Aware Multi-Criteria Indoor Route Planning", World Wide Web Journal (Submitted and Under Review).

3. Salgado, C. (2018, November). "Keyword-aware Skyline Routes Search in Indoor Venues". In Proceedings of the 9th ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness (pp. 25-31). ACM.

4. Chaluka Salgado, Muhammad Aamir Cheema & Tanzima Hashem "Continuous Detour Queries in Indoor Venues", 16th International Symposium on Spatial and Temporal Databases (SSTD) 2019 (Accepted).

5. Salgado, Chaluka, Muhammad Aamir Cheema, and Mohammed Eunus Ali. "Continuous monitoring of range spatial keyword query over moving objects". World Wide Web 21.3 (2018): 687-712.

Thesis including published works declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 3 original papers published in peer reviewed journals or conferences and 2 papers have been submitted and are currently under review. The core theme of the thesis is to study efficient spatial query processing in urban environments. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Faculty of Information Technology, Monash University, under the supervision of Assoc. Professor Muhammad Aamir Cheema and Assoc. Professor David Taniar.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapter 3, 4, 5 and 6 my contribution to the work involved the following:

Thesis Chapter	Publication Title	Status (published, in press, accepted or returned for revision)	Nature and % of student's contribution	Co-auther name(s) Nature and % of co-author's contribution	Co-author(s), Monash student Y/N
3	An efficient approximation algorithm for multi-criteria indoor route planning queries	Published	80% of problem formulation, techniques, algorithm, experiments	Muhammad Aamir Cheema, input into techniques 15%	No
				David Taniar, input into the paper 5%	No
	Category Aware Multi-Criteria Indoor Route Planning	Submitted	80% of problem formulation, techniques, algorithm, experiments	Muhammad Aamir Cheema, input into techniques 15%	No
				David Taniar, input into techniques 5%	No
4	Keyword-aware Skyline Routes Search in Indoor Venues	Published	100% of problem formulation, techniques, algorithm, experiments		
5	Continuous Detour Queries in Indoor Venues	Accepted	80% of problem formulation, techniques, algorithm, experiments	Muhammad Aamir Cheema, input into techniques 10%	No
				Tanzima Hashem, input into techniques 10%	No
6	Continuous monitoring of range spatial keyword query over moving objects	Published	80% of problem formulation, techniques, algorithm, experiments	Muhammad Aamir Cheema, input into techniques 10%	No
				Mohammed Eunus Ali, input into techniques 10%	No

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student name: M G Gaminda Chaluka Ruwanga Salgado

Student signature: Date: July 29, 2019

The undersigned hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor name: Muhammad Aamir Cheema

Main Supervisor signature: Date: July 29, 2019

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Assoc. Prof. Muhamamd Aamir Cheema for the continuous support throughout this incredible journey. His guidance with immense knowledge helped me in all the time of research and writing thesis. I still can remember how patiently he listened to me and motivated me when I'm in desperate situations. I could not have imagined having a better advisor and mentor for my PhD study. Besides my advisor, I would like to thank my co-supervisor Assoc. Prof. David Taniar for his insightful feedback and encouragement.

This research work would not have been possible without the financial support of Monash University. Also, I would like express my gratitude to the staff of the Faculty of Information Technology for the arrangements and support during my PhD study.

I thank my fellow research mates, Ammar Sohali, Arif Hidayat, Agnes Haryanto, Nasser Allheeib, Tenindra Abeywickrama, Utari Wijayanti and Zhou Shou, for their stimulating discussions and productive critics, especially in group presentations, have provided new ideas and directions to the work. Also, thanks a lot for the wonderful memories and for the last minute favors.

Nevertheless, I would like to thank my parents, my brother and my friends for providing me with unfailing support and continuous encouragement throughout my life. Especially, to my father who believed in me. Last but not the least, I would like to thank my beloved wife for supporting me spiritually throughout writing this thesis and my life in general.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Location-based services (LBSs) provide tailored information for users with respect to the location of the user. Some of the common services are navigation systems, emergency services, local advertisement, asset management, entertainment, and social networking. With the increasing availability of location sensitive mobile devices such as smartphones and the evolution of telecommunication technologies, we have witnessed rapid advances in LBSs recently. Unlike other traditional geographic information systems (GIS) and web-based applications, LBSs are more focused on the dynamic and mobile environments [5] in which these services can be used on-the-go. Due to the growing ubiquity of smartphones, these services have been utilized in a variety of people's daily life activities and have become the cornerstone of the mobile experience. According to the Australian Communication and Media Authority (ACMA) [6], 72% of Australians accessing the internet via their mobile phone use an LBS at least once a week.

The increasing use of LBSs has led most of the companies to integrate LBSs into their mobile applications as it has a remarkable impact on their

businesses. For example, retailers can increase their customers' interaction by providing information about their products and promotions. Meanwhile, they can also use LBSs to gain insights on customers' shopping behavior to initiate promotion campaigns or tailor their market strategies. Moreover, the recent surveys report that the majority of trending mobile applications could not have gained such popularity in the absence of the LBSs [7, 8].

LBSs have received significant research attention in the past, and numerous spatial queries have been studied to support various LBSs. For example, k nearest neighbor queries [9, 10, 11, 12, 13, 14], range queries [15, 16, 17, 18] , reverse k nearest neighbor queries [19, 20, 21, 22], spatial keyword queries [23, 24, 25, 26, 27], skyline queries [28, 29, 30, 31, 32], route planning queries [33, 34, 35, 36, 37], detour queries [38, 2, 39, 40, 3], etc. In this thesis, we study a variety of location-based queries in urban environments which typically have a high density of built structures such as shopping malls, airports, hospitals, offices, and transportation networks. An urban environment consists of indoor space and also outdoor space. But, we mainly focus on spatial query processing in indoor venues due to the following two reasons. Firstly, the research [41, 42] has shown that people spend more than 85% of their daily lives in the indoor space such as office buildings, shopping centers, libraries, etc. Thus, it is important to support LBSs in the indoor environments to realize a wide variety of real-world applications such as emergency services, assisted healthcare systems, indoor asset tracking and event planning. For example, directing people to safe exits during emergency evacuations, tracking and monitoring assets used in an airport and providing shopping assistance for customers. Secondly, even though the spatial query processing has been studied extensively in outdoor space (e.g., transportation networks and Euclidean space), these techniques are not suitable for indoor venues as they do not take

into account the unique properties of the indoor space such as hallways, stairs, escalators, rooms, etc. It was shown in [1] that a straightforward extension of the outdoor techniques is up to five orders of magnitude slower than the techniques that are specifically designed for indoor environment exploiting its unique properties. More details are discussed in Section 1.2. This motivates us to study spatial query processing techniques specifically designed for indoor venues.

1.1 Some Popular Spatial Queries

In this section, we briefly describe the spatial queries which are related to this thesis.

Route Planning Queries

A traditional route planning query returns the optimal route from a given source point to a target point that passes through at least one point of interest from each given category. Figure 1.1 shows an example of a network with restaurants, supermarkets and gas stations that are denoted by r_i, m_i and g_i respectively. Assume a user who is traveling from her workplace s to home t and wants to visit a gas station, a supermarket and a restaurant on her way. She can pose a route planning query to find the shortest route that goes through each of such places on her way home. As Figure 1.1 shows, the routes $R_a = \{s \rightarrow r_1 \rightarrow m_3 \rightarrow g_2 \rightarrow t\}$ (solid red line) and $R_b = \{s \rightarrow g_1 \rightarrow m_1 \rightarrow r_2 \rightarrow t\}$ (solid blue line) are two possible routes with the distances (in Manhattan distance) 16 and 20 respectively. The route R_a is returned since it is shortest route among all the possible routes. In the past, many variants of route planning queries have been studied. We provide a comprehensive

overview in Section 2.

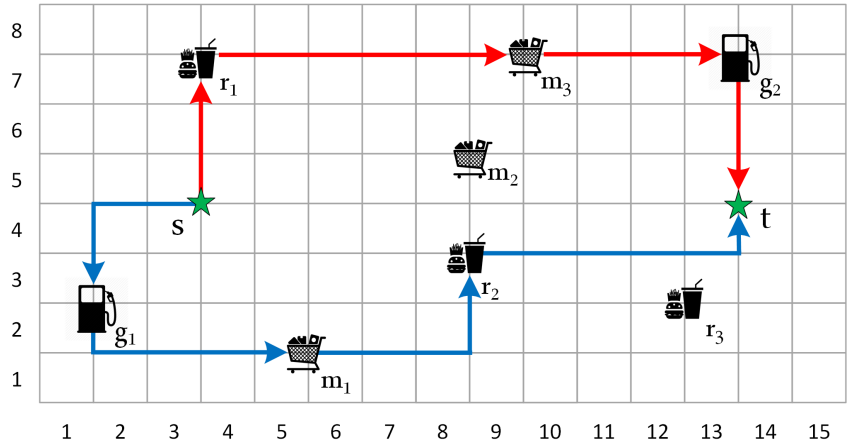


Figure 1.1: Example of a route planning query

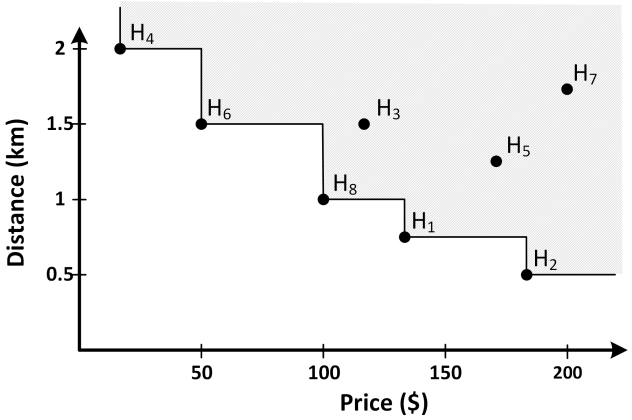
Skyline Queries

Given a set of points P with multiple attributes, a skyline query returns each point $p \in P$ that is not *dominated* by any other point. These points are known as skyline points. A point $x \in P$ dominates another point $y \in P$ if x is better than y in at least one attribute and at least as good as y in all other attributes.

The skyline queries are essential for real-world applications that involve multi-criteria decision making. For example, assume a student who is going to a conference wants to find an accommodation that is cheap and close to the conference venue. Figure 1.2(a) shows a table that lists the distance to the conference venue (in kilometers) and the price per night (in dollars) of each hotel in the city where the conference is held. Figure 1.2(b) shows the data points that correspond to the hotels with two attributes: distance and price. Lets consider the point H_3 and H_8. The point H_8 dominates the point H_3 as both price and distance of H_8 are smaller than those of H_3. Thus, the point H_8 is a better choice than the point H_3, i.e., H_8 dominates H_3. The points H_1, H_2, H_4, H_6 and H_8 (the points which are connected in the figure)

Hotel	Distance (km)	Price ($)
H_1	0.7	140
H_2	0.5	180
H_3	1.5	110
H_4	2	20
H_5	1.3	170
H_6	1.5	50
H_7	1.7	200
H_8	1	100

(a)



(b)

Figure 1.2: Example of a skyline query

are the skyline points as they are not dominated by any other point. Therefore, the hotels that are represented by these points are important in terms of the distance and price because for each hotel, there is no other hotel that is better than it in terms of both price and distance. These skyline hotels are sent to the user as a set of shortlisted hotels and she can make a decision to finalize a hotel of her choice.

As stated in the previous section, traditional route planning queries return a route that optimizes a single criterion, e.g., the shortest route. If the user has multiple criteria (e.g., total distance of the route and total price of the route), a skyline route planning query returns all routes that are not dominated by any other route. More details are discussed in Chapter 4.

Detour Queries

A detour query is a special case of route planning queries where the user specifies only one point that she wants to pass through on her way to the target. Specifically, given a source s and a target t and a set of points P, a detour query returns the shortest route from s to t that passes through at least one point of P. For example, a user shopping in a supermarket may want to go to her car in the car park but may want to pass through a coffee shop on her way out. She may issue a detour query that returns the shortest path which deviates from the original route (may be the preferred path or the shortest path), passes through a point-of-interest, e.g., a coffee, and reaches her destination. Figure 1.3 shows an example of a detour query where the dotted (red) line represents the shortest path while the solid (green) line represents the detour path. We study continuous detour queries in this thesis. More details are presented in Chapter 5.

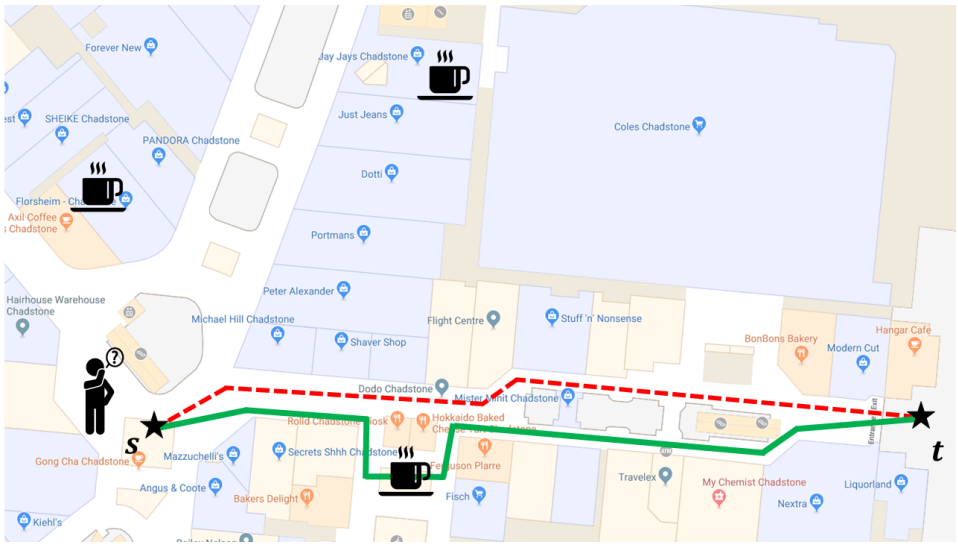


Figure 1.3: Example of a detour query

Spatial Keyword Queries

The spatial keyword queries are one of the most popular queries in location-based services. Given a set of geo-textual objects O where each object $o \in O$ is described by its location and a set of keywords, a spatial keyword query q finds objects that satisfy the requirements of the query in terms of both spatial proximity and textual similarity. For example, Figure 1.4 shows 6 restaurants $R_1 - R_6$. The popular dishes of the restaurants are mentioned below each restaurant. Consider a user may want to find two nearby restaurants that are popular for seafood and steak. She can pose a spatial keyword query with her preferences as keywords "seafood" and "steak". It will return R_6 and R_3 restaurants as these are the two closest restaurants that contain at least one keyword. A wide variety of spatial keyword queries have been studied in the past including range query, k nearest neighbor query, top-k query, and publish-subscribe query. A comprehensive study of these queries can be found in [43].

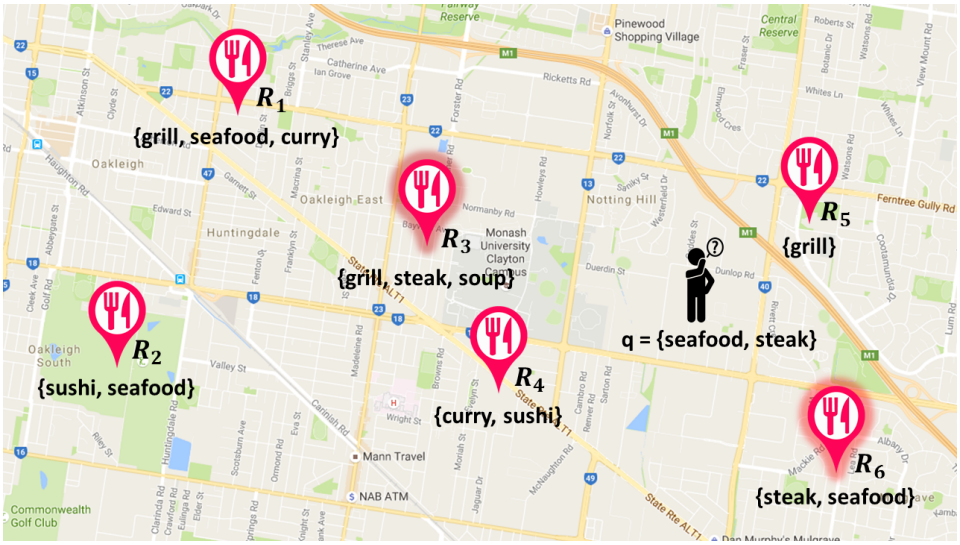


Figure 1.4: Example of a spatial keyword query

1.2 Various Problem Settings for Spatial Queries

As we mentioned earlier, a plethora of studies that investigate the spatial query processing can be found in the literature. These queries are studied using various problem settings. In this section, we briefly discuss some of the well-known problem settings that are used in spatial query processing.

Snapshot vs Continuous

A Snapshot query computes the results of the query only once. For example, a user may want to find the coffee shops within 2 kilometers from her apartment. She may issue a snapshot range query with the range as 2 kilometers and the query location as her apartment. Hence, all the coffee shops within the given range are returned.

In contrast to a snapshot query which is a one-time query processing, a continuous query must keep the query result up-to-date until the query is terminated since the underlying data may change with time. For example, a person who is driving a car may want to find the coffee shops within 2 kilometers from his current location. Hence, the query results must be continuously monitored since his current location may change as the car moves. This example describes a scenario where only the query object is continuously moving. But there are real-world scenarios where both query objects and data objects may be continuously moving. For example, a moving armored car in a battlefield wants to continually find the nearest tank until it reaches the base camp.

Euclidean Space vs Spatial Networks

The spatial distance is an essential feature in the spatial query processing. Methods that are used to measure the spatial distance vary depending on the applications. Euclidean distance, Manhattan distance and network distance are the most common variations of the distance functions used in recent studies. For example, a person walking on a street may want to find the closest Chinese restaurant. In such a scenario, the network distance is more plausible than the Euclidean distance. Hence, the spatial query must take into account the network distance between the query object, i.e., the user, and the data objects, i.e., the Chinese restaurants, to determine the query result. Similarly, a fighter pilot who wants to find the nearby enemy targets may issue a range query that uses the Euclidean distance in measuring the distances.

Outdoor Space vs Indoor Space

As we mentioned earlier, the spatial queries on outdoor space either use Euclidean space or road network depending on the application. The road network is represented using a network graph where the roads and road junctions are represented using the edges and vertices in the graph appropriately. Similarly, the indoor space can also represented using graphs such as the door-to-door graph [44], where doors are represented using vertices while edges represent the connectivity between the doors. However, a straightforward extension of the existing outdoor techniques results in below par performance as it relies on the properties of road networks and fails to exploit the unique properties of indoor space. [1] provides evidence that clearly shows the uniqueness of the indoor space. It reports that the road networks have much lower average out-degree (2 to 4) as compared to the indoor graphs which have average out-

degree up to 400. Therefore, the indoor graphs are much larger relative to the actual area it covers. For example, the indoor graph corresponds to an indoor venue (i.e., Clayton campus of Monash University) which is used in [1] consists of 6.7 million edges and around 41,000 vertices. Compared to this, the road network corresponds to California state consists of around 4.6 million edges and 1.9 million vertices. Therefore, unique techniques that carefully exploit these indoor properties are necessary to provide efficient results.

1.3 Contributions

This thesis aims to study efficient spatial query processing in urban environments. We investigate the limitations of previous studies and propose efficient techniques to answer several spatial queries focusing on the indoor space. Below, we briefly discuss the contributions made in this thesis.

1.3.1 Category Aware Multi-criteria Indoor Route Planning Queries

A route planning query has many real-world applications and has been studied extensively in outdoor spaces such as road networks or Euclidean space. Despite its many applications in indoor venues (e.g., shopping centres), almost all existing studies are specifically designed for outdoor spaces and do not take into account the unique properties of the indoor spaces. Hence, we study the problem of category aware multi-criteria route planning query in indoor space, denoted by CAM, which returns the optimal route from an indoor source point to an indoor target point that passes through at least one indoor point from each given category while minimizing the total cost of the route. Although we show that CAM query is NP-hard, the exact solutions may be feasible for

the scenarios when the number of query categories is small, which is typically the case in many real-world applications. However, when the number of query categories increases, the exact solutions become increasingly expensive. Thus, based on a novel *dominance-based* pruning, we propose an efficient approximation algorithm that generates high-quality results. We provide an extensive experimental study conducted on a real-world dataset and demonstrate that our algorithms are efficient and produce high-quality results.

The results were published in *ACM International Conference on Advances in Geographic Information Systems* (ACM SIGSPATIAL) 2018. Additionally, an extended version has been submitted to the *World Wide Web Journal* and it is currently under review.

1.3.2 Keyword-aware Skyline Routes Queries in Indoor Venues

We study an interesting route planning problem called keyword-aware skyline routes (KSR) query which returns a set of *non-dominated* routes, i.e., skyline routes, instead of an optimal route. We take into account two attributes, route distance and the number of shops/stores that are visited, to determine the dominance of a route over another route. A route R_x dominates another route R_y if R_x is better than R_y in at least one attribute and at least as good as R_y in all other attributes. For example, let R_x and R_y be two routes that start at a point s and end at a point t, covering all the user given keywords. Let the distance of route R_x and R_y be 100 and 150 meters respectively. Assume that route R_x passes through only one shop while route R_y goes through three different shops. Thus, route R_x dominates route R_y since both route distance and number of visited shops/stores of route R_y are smaller than those of route

R_x. KSR queries facilitate the users to find the most promising routes among the all possible routes, with respect to these attributes. Although we prove that the problem of KSR query is NP-hard, we propose an efficient exact solution for the case when the number of query keywords is small which is typically the case in real-world applications. We present an extensive experimental study on a large real-world shopping center containing real products and show the efficiency of the proposed techniques.

This work was published in *Ninth ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness* (ISA) 2018.

1.3.3 Continuous Detour Queries in Indoor Venues

We study continuous detour queries in the indoor space. A continuous detour query finds the nearest indoor detour object (e.g., an ATM) for a moving user who is walking towards a target location in an indoor venue, where the detour distance of an indoor object is measured as the total indoor distance of the object from the user's current location to the detour object and from the detour object to the target location. The continuous detour query has been already studied in the outdoor space [3, 45], but the solutions are not adaptable for the indoor space due to the unique characteristics of indoor venues. We develop the first solution for the efficient processing of the continuous detour query in the indoor space. The novelty of our solution comes from the computation of the safe zones for the indoor objects by exploiting the geometric properties of the hyperbolas and the uniqueness of the indoor space. A safe zone for an indoor object represents an area where it is guaranteed that the indoor object remains the nearest detour with respect to a partition door for a user moving towards a fixed target. The key ideas behind the efficiency of our solution are reducing the number of re-evaluation of the detour queries for the location

change of a moving user, precomputing the safe zones, and indexing them using a grid structure.

The results of this work have been submitted to *International Symposium on Spatial and Temporal Databases* (SSTD) 2019 and the paper is currently under review.

1.3.4 Continuous Range Spatial Keyword Queries over Moving Spatio-textual Objects

We propose an efficient solution for processing continuous range spatial keyword queries over moving spatio-textual objects (namely, *CRSK-mo* queries). In *CRSK-mo* queries, the relevance of an object to a query is determined in terms of both spatial proximity and textual similarity between the query and object. Hence, every object with the relevance score less than or equal to a user given threshold score is returned as the answer. Major challenges in efficient processing of CRSK-mo queries are as follows: (i) the query range, i.e., threshold score, is determined based on both spatial proximity and textual similarity; thus a straightforward spatial proximity based pruning of the search space is not applicable as any object far from a query location with a high textual similarity score can still be the answer (and vice versa), (ii) frequent location updates may invalidate a query result, and thus require frequent re-computing of the result set for any object updates. To address these challenges, the key idea of our approach is to exploit the spatial and textual upper bounds between queries and objects to form *safe zones* (at the client-side) and *buffer regions* (at the server-side), and then use these bounds to quickly prune objects and queries through smart in-memory data structures. We conduct extensive experiments with a real-world dataset that verify the effectiveness

and efficiency of our proposed algorithm.

For the ease of presentation, we present our techniques for CRSk-mo queries in Euclidean space. However, we also show how the techniques can be easily extended for indoor venues. The results have been published in the *World Wide Web Journal* 2018.

1.4 Thesis Organization

Below, we present the structure of the rest of the thesis.

- Chapter 2 presents a review of the related work.
- Chapter 3 describes our work on category-aware multi-criteria indoor route planning queries.
- Chapter 4 covers our proposed techniques to solve keyword-aware skyline routes search in indoor venues.
- Chapter 5 constitutes our work on continuous detour queries in indoor venues.
- Chapter 6 covers our work on continuous monitoring of range spatial keyword query over moving objects.
- Chapter 7 concludes our research and describes several possible directions for future work.

Chapter 2

Literature Review

This chapter provides a brief overview of the related work for each type of queries we study in this thesis. More specifically, in Section 2.1, we review the existing techniques to answer spatial queries in indoor space. We provide an overview of the related techniques for route planning queries in Section 2.2. We present the related work on skyline queries in Section 2.3 followed by an insight into the indexing and query processing techniques of continuous queries in Section 2.4. In Section 2.5, we provide a detailed discussion of the existing techniques of detour queries. Finally, we survey the related work on spatial keyword queries in Section 2.6.

2.1 Query Processing in Indoor Space

The query processing in indoor space has received considerable attention in recent years where several query processing and indexing techniques were proposed. A comprehensive taxonomy for querying indoor data, shortest distance/path, range, and k nearest neighbor queries under various settings can be found in [46, 47, 48, 49]. RTR-Tree and TP^{2}R-tree [50] are extensions of

R-tree to index trajectories of indoor moving objects. Xie et al. [51] develop a composite indexing structure *ind*R-tree that indexes indoor entities into different layers, namely geometric, topological and object layers.

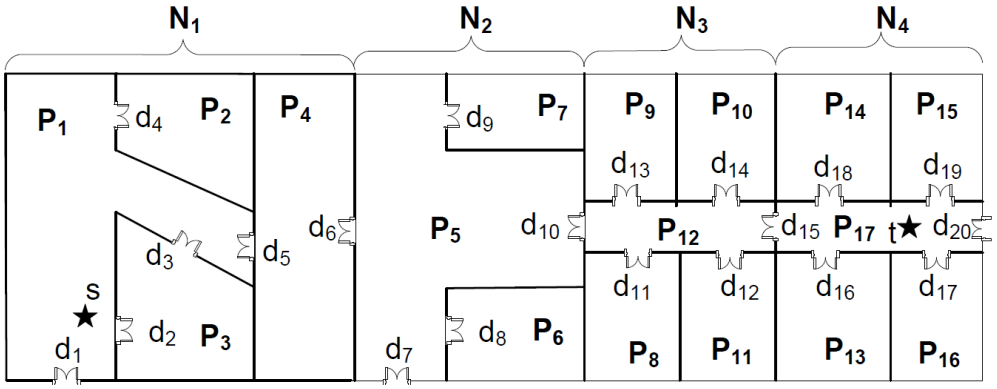


Figure 2.1: Example of an indoor venue [1]

Yang et al. [52] propose accessibility base (AB) graph that describes the topology of a floor plan of an indoor venue by capturing the underlying connectivity and accessibility of the particular indoor space. Figure 2.1 illustrates an example of an indoor venue that consists of 17 indoor partitions (P_1 to P_{17}) and 20 doors (d_1 to d_{20}). The AB graph corresponds to this indoor venue is shown in Figure 2.2. As Figure 2.2 depicts, a vertex represents an indoor partition while an edge captures the connectivity between two indoor partitions. For example, there is an edge with label d_4 between vertex P_1 and P_2 in the AB graph since the indoor partitions P_1 and P_2 are connected by door d_4 (See Figure 2.1). Although the AB graphs capture the connectivity information, they do not support indoor distance computations. To handle this problem, distance matrix [52] is introduced. It stores door-to-door indoor minimum walking distances. Even though this approach is optimal in retrieving the distance between any two doors, the storage and pre-processing costs do not scale well for large indoor venues.

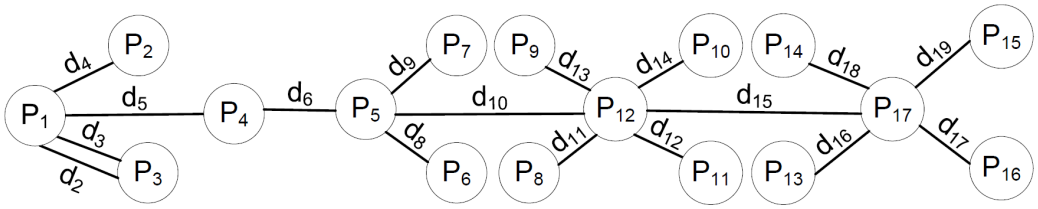


Figure 2.2: Example of an AB graph [1]

The door-to-door (D2D) graph [44] is one of the most notable techniques which has been used in most of the studies in the literature since it enables various query processing techniques in road networks [53, 54, 55, 56, 57, 58] to be applied in the indoor space. Figure 2.3 shows the corresponding D2D graph for the indoor venue which is shown in Figure 2.1. As Figure 2.3 shows, the vertices represent the doors in the indoor space. A weighted edge between two vertices is created if they are connected to the same indoor partition (e.g., room, hallway, etc.) where the edge weight is the indoor distance between the corresponding doors. For example, the doors from d_1 to d_5 are connected to each as they all belong to the same partition, i.e., P_1.

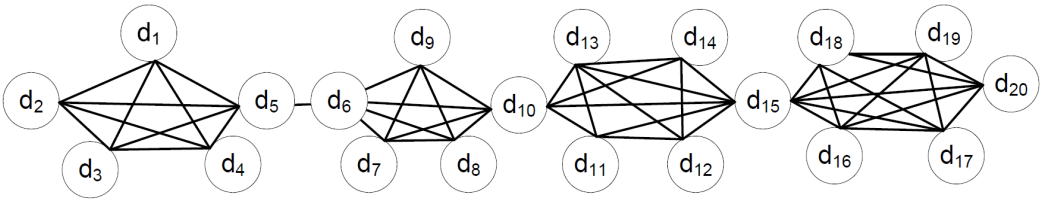


Figure 2.3: Example of a D2D Graph [1]

Shao et al. [1] introduce an efficient index structure called IP-tree that takes into account the unique properties of indoor space in building the index. In the IP-tree, adjacent indoor partitions (e.g., rooms, hallways, staircases, etc.) are combined to form leaf nodes. Then the adjacent leaf nodes are combined to form intermediate nodes. This process is iteratively continued until all nodes are combined into a single node (i.e., root node). Figure 2.4

shows the corresponding IP-Tree for the indoor venue which is shown in Figure 2.1. As Figure 2.4 illustrates, the indoor space is first converted into four leaf nodes (N_1 to N_4). Each leaf node consists of several indoor partitions, specifically $N_1 = \{P_1, \ldots, P_4\}$, $N_2 = \{P_5, \ldots, P_7\}$, $N_3 = \{P_8, \ldots, P_{12}\}$ and $N_4 = \{P_{13}, \ldots, P_{17}\}$. Then, the leaf nodes are iteratively merged until the root node is formed. For example, N_1 and N_2 are merged to form N_5, whereas N_3 and N_4 are merged to form N_6. For each node, a distance matrix is created to store the distances between every access door and every door in the particular node. VIP-Tree or Vivid IP-Tree [1] is an improvement of the IP-tree. For each door d_i, it stores the following additional information. Let N be a leaf node that contains door d_i. For every door d_j which is an access door in one of the ancestor nodes of N, VIP-tree stores $dist(d_i, d_j)$ as well the next-hop door d_k on the shortest path from d_i to d_j. This information can be efficiently computed by the efficient shortest distance/path algorithms using an IP-tree. Compared to the existing indexing techniques, VIP-tree has demonstrated more efficiency and higher scalability. In this thesis, we utilize VIP-tree to index the indoor space.

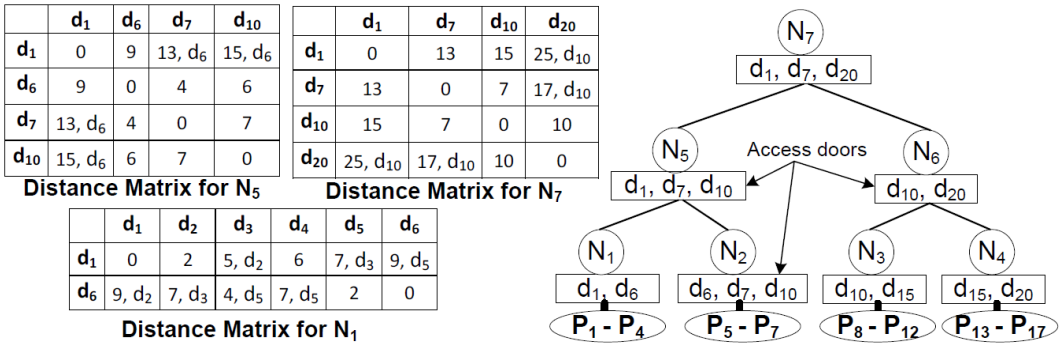


Figure 2.4: Example of a VIP-tree [1]

2.2 Route Planning Queries

A large body of research has been done on developing techniques to efficiently process route planning queries. Trip planning query (TPQ) [33] returns the shortest route starts at a source location, passes through at least one object from each given category and ends at a target location. For example, Figure 2.5 shows a network with three different categories of point sets, particularly green, orange and blue circles represent gas stations (denoted by g_i), banks (denoted by b_i) and pharmacies (denoted by p_i) respectively. Moreover, source and target points are shown in star shapes. Consider a user who is traveling from the point s to the point t wants to go to a bank, a gas station and a pharmacy on her way. She may use a TPQ to obtain the route $R = \{s \rightarrow p_3 \rightarrow b_2 \rightarrow g_3 \rightarrow t\}$ (the route that is shown in solid line) which is the shortest route that covers all given categories. They prove that the problem of solving TPQ is NP-hard and

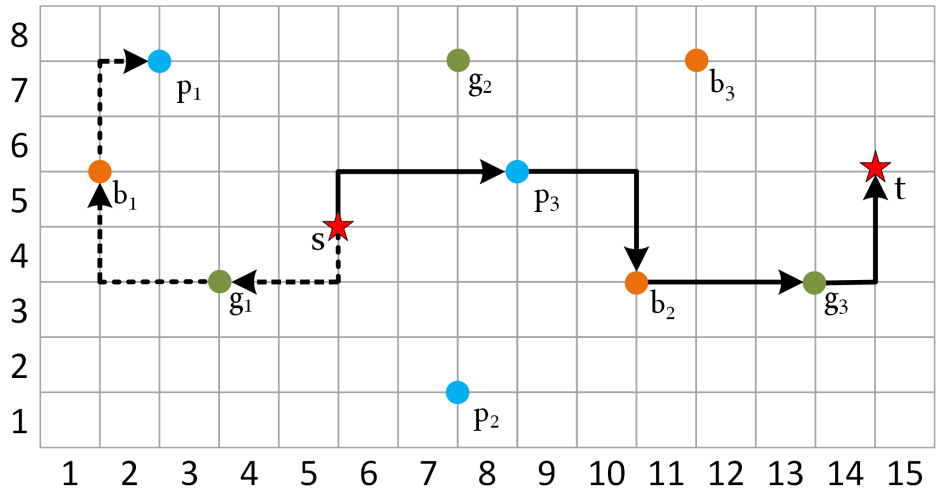


Figure 2.5: A network with three different categories of point sets

propose two near-optimal solutions, namely nearest neighbor algorithm and minimum distance algorithm. These two algorithms are based on triangular inequality property of the metric space. The nearest neighbor algorithm starts

from the source vertex and incrementally adds the nearest neighbor of the last vertex to the trip from the vertices that belong to the categories which have not been covered yet. The minimum distance algorithm first finds a set of candidate vertices by obtaining a vertex per given category with the minimum trip distance from the starting point to the target point via the particular vertex. Then, the trip is formed by iteratively visiting the nearest neighbors from this set of candidate vertices.

Sharifzadeh et al. [34] introduce a variant of TPQ called optimal sequenced route (OSR) query that takes into account an order of visiting places which is given by the user. For example, assume a user may want to go to a gas station first, and then to a bank and finally to a pharmacy. Therefore, the user can use an OSR query to get an optimal route in terms of traveling distance. As Figure 2.5 shows, the OSR query returns the route $R = \{s \rightarrow g_1 \rightarrow b_1 \rightarrow p_1\}$ (the route that is shown in dotted line) as the result since it is the shortest route covering all the categories in the given sequence. In order to solve OSR in Euclidean space, they present two exact algorithms light optimal route discoverer (LOAD) and R-tree based LORD (R-LOAD) which is an improvement of the LORD. They also propose a solution called progressive neighbor exploration (PNE) to handle OSR queries in the metric space. The idea behind the PNE is to incrementally generate a set of candidate routes for the given category sequence. They iteratively add the nearest neighbor vertices to a partial candidate route from the vertices that has not been visited. They maintain the given sequence when adding a new vertex to a route. Also, they add the next nearest neighbor of the end vertex of the previous route since it may produce a better route later. For example, the PNE starts from the point s and it adds the nearest neighbor belongs to the category g (first category in the given sequence), i.e., the point g_1 to the route. Then it adds the point b_1

(which belongs to the next category in the given sequence) to the end of the current route since it is the nearest neighbor of the point g_1. Meanwhile, it also generates another route from the point s via the point g_2 since it is the next nearest neighbor of point s. They use a min-heap to store these candidate routes based on their route distances. Hence, the route with the minimum distance is popped from the heap at each subsequent iteration and expanded further. There are several works in the literature [59, 60, 61, 62, 63] that study OSR queries.

Cao et al. [35] introduce another variant of route planning query called keyword-aware optimal route (KOR) search, which covers all the user given keywords while satisfying a user-specified budget constraint and optimizing objective score of the route. They show that the problem of answering the KOR query is NP-hard. Hence, they propose two approximation algorithms, specifically OSScaling and BucketBound, with proven approximation bounds. The basic idea of OSScaling algorithm is to iteratively generate partial routes from the best one among the already found partial routes. The BucketBound algorithm uses buckets to organize the partial routes based on their best possible objective scores. Hence, the best partial route is determined efficiently in each iteration. Zeng et al. [36] find an optimal route such that the keyword coverage is maximized without exceeding a given budget constraint. The purpose of such a route is to optimally satisfy the user's weighted preferences. They define an admissible heuristic exploiting the submodular property where the optimal route is obtained using a variant of A* algorithm. Chen et al. [64] study a new type of route planning query called multi-rule partial sequenced route query in which the users set traveling preferences/restrictions when they issue a query. First, the given preferences/restrictions are evaluate to determine a set of traveling rules by employing the topological sort. Then,

an optimal route that satisfies all these traveling rules is obtained utilizing a heuristic approach. They propose three different heuristic approaches that return optimal routes which are very close to the shortest routes.

Yao et al. [37] study another variant of route planning query called multi-approximate-keyword routing (MAKR) query. A MAKR query finds a route with the shortest length such that it covers at least one matching object per given keyword while satisfying the string similarity constraints. They propose an exact solution called progressive path expansion and refinement (PER) algorithm that generates partial candidate paths and refines them until the optimal path is determined. It starts with the shortest path from the source point to the target point and progressively adds a point whose string is similar to one of the given query keywords, to the current partial candidate path while minimizing the route distance. Accordingly, it generates all possible candidate paths until the shortest path that covers all keywords is determined. Moreover, the distances of partial paths are estimated using pre-computed landmarks to reduce the cost of the shortest path distance computations. Their exact solution is efficient and works well for less number of query keywords since the MARK problem is NP-hard. To support the queries with a large number of query keywords, they present an approximate solution called global minimum path (GMP) algorithm. First, for each query keyword, the GMP algorithm finds the closest point with respect to both source and target points. Then, starting from the source point, it keeps adding the nearest neighbor point from this set of candidate points until all the keywords are covered.

Shao et al. [65] are the first to study the indoor trip planning queries. They propose an exact solution called VIP-tree neighbor expansion (VNE). The main idea of VNE is to explore the nearest neighbors one-by-one while adding them to the end of the partial candidate routes. Once such a route is

constructed, the current nearest neighbor of the second last node of the route is replaced with the next nearest neighbor to assure that all possible partial candidate routes are taken into account. This process is continued until the shortest route that covers all the given categories is found. They introduce a distance pre-computation similar to [1] where the distances between all the access door and the indoor objects (assuming the number of objects in an indoor venue is very small) are stored to reduce the distance computation cost. VNE takes into account the unique properties of the indoor space and features two levels of pruning, particularly at pre-processing phase and at query processing time. In Section 3.3.2, we discuss the limitations of related work in answering the problem that is studied in Chapter 3.

2.3 Skyline Queries

The skyline operator is introduced in [28]. They propose two approaches, block-nested-loop processing and extended divide and conquer approach. Since then, the skyline query processing in databases has drawn a significant attention where an enormous amount of research is found in the literature [66, 29, 30, 67, 68, 69].

Most of these works focus on efficiently finding skyline points in traditional databases. Kossmann et al. [29] propose an online algorithm to process skyline queries. They compute the skyline in a batch using a nearest neighbor approach where they progressively report the skyline points. However, their techniques process an object several times to verify whether it is a dominant or a dominated point. Furthermore, in high dimensional spaces, this approach is more likely to get duplicate points in the resultant skyline. To address this issue, Papadias et al. [30] propose a solution that utilizes an R-tree to index

the data space. They introduce an algorithm called branch-and-bound skyline (BBS) that performs a nearest neighbor search on the R-tree index such that only a single access to each R-tree node (each node contains unique data points) is required to identify the skyline points. Thus, it does not retrieve duplicates like the NN approach in [29]. Lin et al. [70] introduce an online algorithm to compute the skyline for the most recent n elements in a rapid data stream. All these methods focus on skyline query processing in Euclidean space. Next, we present some important studies that apply the skyline operation on spatial database systems, specifically, in road networks.

Tian et al. [71] introduce the concept of skyline paths and propose a novel skyline path search algorithm called SkyPath. Their techniques utilize edge attributes of the road network to determine the skyline paths between a given source and target locations. First, they determine a skyline path whose summation of all attribute values is the smallest compared to other possible paths. Then they greedily add a relay node to the path and check whether the new path is itself a skyline path. This process is continued until all the skyline paths are determined. Kriegel et al. [32] present a best-first-based graph exploration that takes into account route preferences based on arbitrary road attributes. Instead of the naive method used in [71], they used triangle inequality based approach to estimate the minimum attribute values of a path formed through a relay node that is greedily retrieved. They assume that different paths have different values correspond to each attribute. Moreover, they do not take into account any POIs or the distance between them in route construction. Deng et al. [31] study a problem called multi-source skyline query that finds skyline POIs in road networks with respect to the relative network distances to multiple query points. Given a set of m query objects and n data objects, they map each data point to m-dimension space with i^{th} dimension

refers to the road distance between the data object o and the i^{th} query object. They propose three algorithms: Euclidean distance constraint, lower bound constraint and collaborative expansion to answer multi-source skyline queries. Huang et al. [72] study another type of skyline problem in which the domination of points takes into account two attributes, namely road distance to the query and detour distance from a predefined route of the query. Jang et al. [73] are the first to study the problem of continuous skyline queries in road networks. They propose an approach that utilizes the precomputed results to determine the skyline objects. However, their techniques are not applicable in large datasets as their precomputation incurs tremendous processing cost. Also, the query results do not deliver useful information since the skyline points far away from the query object are also included in the query results. Huang et al. [74] introduce a grid index to manage the information of the data objects on the road network such that only a small proportion of data objects are accessed in query processing. First, they determine a set of candidate skyline points in global space by accessing only a few grid cells. Then a set of points where the query results change is determined to support continuous monitoring.

Hsu et al. [75] study a skyline trip planning query that returns a set of skyline travel routes within a user-defined spatial range. They take into account some factors such as the visiting time information of POIs and the distances to the set of query points, when retrieving the travel routes. Aljubayrin et al. [76] study a problem that finds a set of skyline routes passes through multiple POIs covering a given set of categories. They take into account both trip distance and trip aggregated cost in ranking the routes. They prove that the problem is NP-hard and propose two fast algorithms that produce near optimal results in practice. They precompute and store the distances between

POIs and some geographical regions in the network. The proposed algorithms work by repeatedly iterating through two stages, particularly POIs nomination and trip construction stages. In the first stage, the most superior point per given category is determined. Then, in the next stage, all possible routes from the source point to the target point that pass through the set of nominated POIs, are constructed using a greedy approach to determine the skyline routes. However, these techniques are not applicable in answering the problem that is studied in Chapter 4. More details are presented in Section 4.3.2.

2.4 Continuous Queries

Several techniques for indexing moving objects/queries have been proposed in the literature to support continuous query processing. Below are the details. Saltenis et al. [77] propose a novel indexing technique called the time-parameterized R-tree (TPR-tree) which indexes the current and anticipated future positions of moving objects by transforming the location of each object into a linear function of time. Tao et al. [78] develop an improved version of the TPR-tree that utilizes efficient insertion and deletion techniques. But these indexing structures are too expensive to maintain as the known-trajectory assumptions do not support many real-world scenarios as the future locations and velocities of the objects are frequently changed.

Prabhakar et al. [79] introduce a novel strategy call Q-index that indexes the queries instead of the objects. Although it reduces the index maintenance cost, their techniques are limited to range queries. Wu et al. [80] propose a new query indexing method called containment encoded square (CES) based indexing. Kalashnkov et al. [81] propose an R-tree variant called R^{UB} -tree which is an R-tree with update buffering where the proposed techniques are

applicable to any other R-tree variants and various kinds of multi-dimensional indexes (in memory grid-indexes). However, all these techniques impose high communication and computation overhead when they are applied in real-world applications since they focus only on reducing the evaluation cost regardless of the location update cost. An attractive technique called *safe region* (also known as safe zone) is proposed in [82, 79] to overcome these issues. Hu et al. [82] propose a generic framework based on the safe region approach to monitor continuous range and k-nearest neighbor queries. A safe region is computed for each object based on the set of available queries such that the query results are guaranteed to remain the same until the objects lie within their safe regions. For example, Figure 2.6 shows two range queries q_1 and q_2 with range r_1 and r_2 respectively. The object o_1 is a result for both queries since o_1 located within the both query ranges r_1 and r_2. Moreover, the shaded area shows the safe region of object o_1 where the results of two queries do not change until object lies within this area. Consequently, the objects do not send location updates to the server unless they move out of their safe region or the server requests for a location update. The intuition behind the safe regions is to reduce the communication and computation overhead by minimizing the location updates.

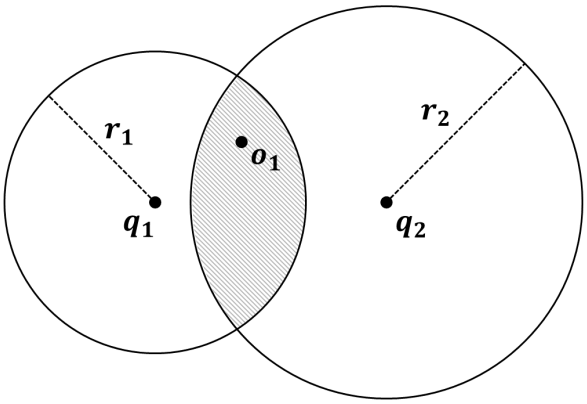


Figure 2.6: Example of a safe region

Cai et al. [83] propose a scalable and adaptive technique called monitoring query management (MQM), which aims to reduce the communication cost and the server workload by assigning a resident domain for each object. The resident domain is computed considering the heterogeneous computational capabilities of moving objects. As opposed to the safe zone, an object (i.e., client device) monitors some selected query boundaries along with its domain boundary. Hence, the object reports its location to the server when it crosses over some query boundary or moves out of its resident domain. In the first case, the server updates the affected query results accordingly while in the second case, the server determines a new resident domain for the particular object. Jung et al. [84] propose query region tree (QR-tree) and also BQR-tree by improving the resident domain concept. Recently, the safe regions for moving circular range queries over stationary objects has been proposed in [85]. However, all these techniques consider only the spatial information and fail to exploit the textual information. Hence, these techniques cannot be used in continuous spatial keyword query processing that is studied in Chapter 6.

The grid index structure is used in several studies that are performed on processing continuous queries over moving objects. Yu et al. [86] propose two approaches based on a grid index to evaluate continuous k-nearest neighbor (CkNN) queries over moving objects. One approach is based on indexing objects while the other one is based on indexing queries. Also, they present an extension (hierarchical grid) to address performance degradation in highly skewed data. Mouratidis et al. [87, 88] present a new grid partitioning technique called conceptual partitioning where space around a cell is partitioned into direction aware conceptual rectangles. Cheema et al. [89] introduce a novel technique called CircularTrip to access data in grid-based indexing structure. The CircularTrip starts from one cell intersected by the query point and access

cells around the query point which are intersected by a circle centered at query point with a determined radius. The conceptual grid tree [4] accesses the grid by considering it as a conceptual tree. Due to the tree based structure, it supports all the spatial query algorithms that can be applied on the R-tree.

Now we present some of the notable research in the literature that study continuous queries in spatial networks. Kolahdouzan and Shahabi [10] propose two approaches, particularly intersection examination (IE) and upper bound algorithm (UBA). The IE approach performs kNN queries at every intersection on the path to determine split points. The split points are the locations on the path at which the results of the moving query change. In contrast to IE, the UBA approach postpones the kNN computations to the locations that are necessary. Cho and Chung [12] propose an approach that performs snapshot kNN queries at each intersection point of the query path. To speed up the kNN computation, they maintain NN results in a set of selected nodes called condensing points. Moreover, they present an improvement of the propose algorithm that further determines invalid points where such kNN computations are unnecessary. Nutanong et al. [90] propose a technique called V*-diagram which is a safe region based technique. Each time they determine x auxiliary objects (where $k < x$), i.e., some extra objects in addition to the kNN objects of the moving query point, to construct the safe zone of the moving kNN query.

2.5 Detour Queries

In-Route Nearest Neighbor (IRNN) queries in [2] assume that the user is sticked to a fixed daily route where she even returns back to the particular route after visiting a detour facility. Hence, they determine the minimum detour point by obtaining the nearest neighbors of the given route. They propose

four approaches to process IRNN queries. More specifically, simple graph-based approach runs the Dijkstra algorithm at each branch point with a path search upper bound to determine the nearest neighbors. Hence, it is more expensive when the number of branch points is large. To address this issue, recursive spatial range query-based (RSR) is proposed. It utilizes a Euclidean search bound to determine whether a path computation from a branch point is necessary since the Euclidean distance provides a lower bound to the network distance. Then, if there is no facility within the Euclidean distance range, intuitively, no network distance based facility exists in the search area. For example, as Figure 2.7 shows, a new path computation for branch point r_4 is unnecessary since no facility is within the Euclidean search bound T''. They propose another approach called spatial distance join based that utilizes spatial distance join operation to tighten up the search bound at the beginning of the query rather progressively determine the bound as in the RSR. The main idea of their final approach called precomputed zone-based is to precompute zones of the road network such that the nearest neighbors can be determined efficiently. Chen et al. [39] introduce Path Nearest neighbor (PNN) which is capable of

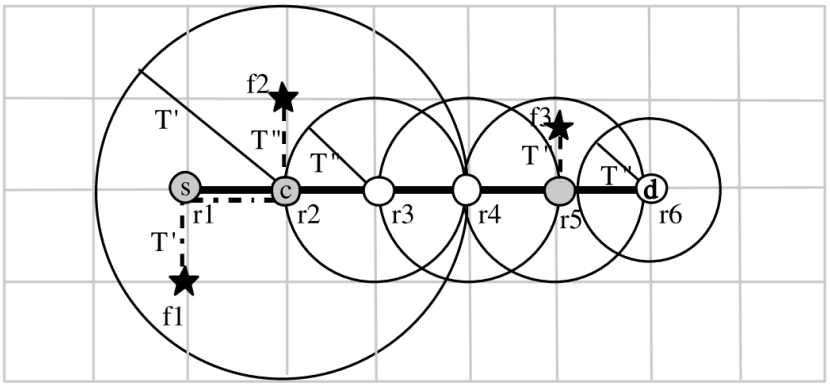


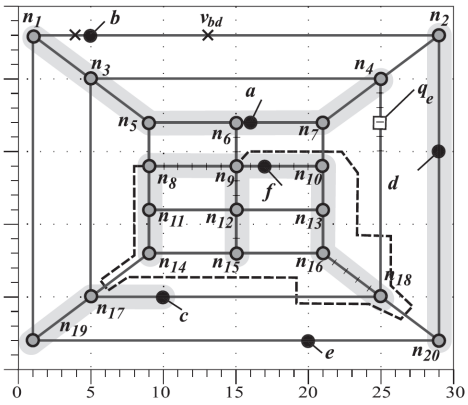
Figure 2.7: Example of RSR approach [2]

monitoring nearest neighbors to the user path, for both cases in which the user

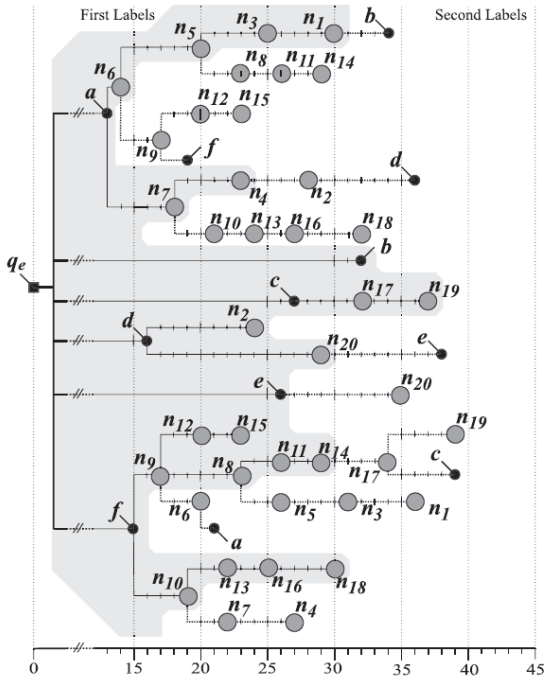
moves along the preferred path and the user deviates from the preferred path. They propose three phase algorithm based on the best first search. Initially, they search for candidate set and then verify the set of candidate nodes that should be visited. Finally, a monitoring phase to maintain the k-PNN results considering the user movements. Basically, they monitor the nearest neighbor for a dynamically changing path rather than a query point. Shang et al. [40] study a detour problem where a preferred path is given along with a detour distance threshold. They initially divide the path into a set of segments and find a set of candidates whose detour distances less than a given threshold. After finding the set of candidates, they compute lower bounds for the detour costs of each candidate to avoid refining all of them. Then, the local best detours are determined by refining each of them. Finally, among all the local best detours, the detour with minimum detour cost is selected as the global best detour.

Nutanong et al. [3] study a continuous detour query that returns k minimum detour objects with respect to user location. They incrementally construct an order-k shortest path tree (SPT) originating from the target point. In an order-k SPT, the branches are overlapped in a way that each node is appeared k times in k different branches. Moreover, labels are created at each node corresponding to these overlaps where such a set of labels accumulates k minimum detour objects with respect to the particular node. Thus, when the query point is at any node, the k minimum detour objects can be easily determined by looking at the set of labels of the particular node. Figure 2.8 shows an example of order-k SPT with k of 2. Note that the Figure 2.8(b) illustrates the order-2 SPT corresponds to the network that is shown in Figure 2.8(a). The set of first labels (i.e., the results of a traditional SPT) is in the shaded area. The two minimum detour objects for node n_6 are objects a and f since

the node n_6 appears first in the branch of a and again in the branch of f later. Furthermore, the result of a query point on an edge $e(n_i, n_j)$, is determined by



(a)



(b)

Figure 2.8: Exmaple of (a) a network and (b) an order-2 SPT [3]

taking into account the results of nodes n_i and n_j, and the detour objects along the particular edge. [45] investigates detour queries in obstructed space. They propose an approach that constructs safe zones by retrieving $k + x$ $(x > 1)$ nearest obstructed detour objects with respect to the user location where k is

the number of detour objects that is monitored by the query.

A variant of nearest neighbor query called aggregated nearest neighbor (ANN) query can be also utilized to answer detour snapshot queries. Yiu et al. [13] propose three techniques that utilize Euclidean distance bounds, spatial access methods and network distance materialization structures. [91, 92] propose two approaches based on the network Voronoi diagrams. Each of these approaches consists of two phases, namely, searching phase and pruning phase. In searching phase, they continually search for the next nearest neighbor from each query point until a common object is found. After a candidate set is obtained, they continually expand the search by computing the next nearest neighbor for a certain query point while pruning unqualified objects. Zhang et al. [14] study a variant of ANN query that takes into account both spatial proximity and textual similarity. They propose an indexing schema called dual-granularity where bitmaps are integrated into a G-tree. They consider POIs on the edges as newly added vertices in the network graph. Thus, it may expand the scale of the original graph by several times when it is applied in the indoor space, thereby leading to high space and query overheads. However, we find these techniques are either having different aims or not applicable in answering the problem which is studied in Chapter 5. We present more details in Section 5.3.2.

2.6 Spatial Keyword Queries

There is a vast body of research aimed at developing efficient query processing techniques for spatial keyword queries. A spatial keyword query returns relevant contents with respect to given spatial and textual arguments. Recently, four types of spatial keyword queries have received a special attention, namely

Boolean kNN query, top-k kNN query, top-k range query and Boolean range query (See [43] for a nice survey).

Focusing on the evolution of indexing structures, Zhou et al. [23] propose a hybrid index structure that integrates inverted indexes [93] and R*-trees [94]. Felipe et al. [95] propose an index structure called information retrieval R-tree (IR^2-tree) that combines signature files with an R-tree. [96] introduces information R-Tree (IR-tree) which is similar to IR^2-tree. In this index, inverted files are used instead of signature files. Li et al. [97] propose an index which is also called IR-tree that stores only one inverted file for all the nodes. Moreover, there are several variants of IR-tree exist in the literature, such as WIR-tree [98], CIR-tree and CDIR-tree [96]. WIR-tree is constructed in a bottom-up manner where objects are grouped using a word partitioning algorithm to form leaf nodes. Then the keyword-union and spatial properties are taken into account in constructing the next level of the tree. CIR-tree and CDIR-tree consist of inverted files for text retrieval and an R-tree to determine spatial proximity. These indexes consider both textual and spatial attributes to prune the search space at query time. Rocha-Junior et al. [99] propose spatial inverted index (S2I) based on inverted R-trees where the set of objects for each term/keyword is stored in a distinct aggregated R-tree (i.e., aR-tree). Vail et al. [24] propose two spatial-textual indexing schemes, specifically spatial primary index (ST) and text primary index (TS) that are based on the grid index structure. [100, 101] also use an inverted grid index structure to organize the related objects for each keyword. The spatial keyword inverted file (SKIF) [102] employs an inverted file that is capable of representing both spatial and textual information. It partitions the working space into a number of grid cells and each cell is treated similar to a textual keyword as opposed to the previous studies. Zhang et al. [25] present integrated inverted index (I^3)

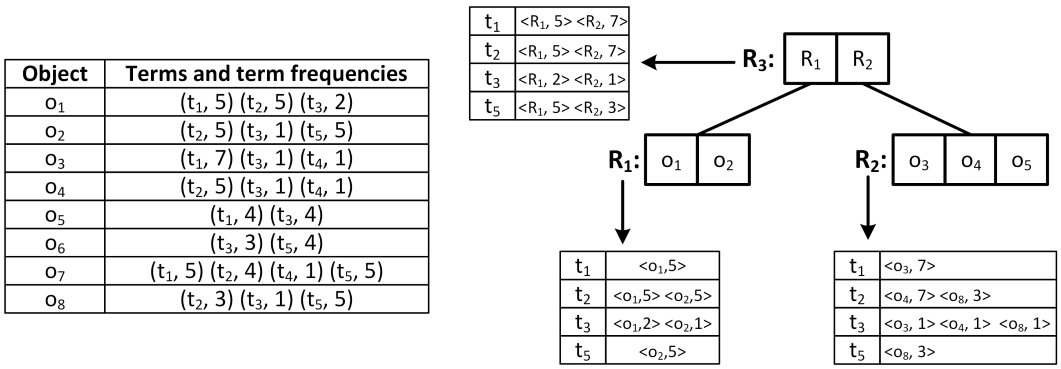


Figure 2.9: Example of an IR-tree and its inverted files

that integrates lists of inverted files to a quad-tree. ILQ-tree [103] is another quad-tree based index structure with an inverted index, which is designed to exploit both spatial and keyword based pruning techniques to reduce the search space in query time. Since all these studies do not look into continuous query processing, we find no sensible way of extending these indexing structures to support the problem that is studied in Chapter 6.

Moreover, [26] and [104] investigate moving top-k spatial keyword queries over stationary objects. Similar to other traditional approaches in moving query processing, the safe zone concept is used to continuously monitor the query results. But their approaches are varied from other approaches since both spatial and textual aspects are considered in computing the safe zones. [26] proposes an algorithm that constructs multiplicatively weighted Voronoi (MW-Voronoi) diagram to determine the safe zones. In [104], the dominant zones for objects are identified and then those regions are utilized to compute the safe zones. These works focus on continuously monitoring the results of moving queries. Wang et al. [27] introduce a novel adaptive index called AP-tree which groups the registered queries with respect to their textual and spatial properties. They propose two techniques, keyword partition and spatial partition that recursively divide the queries in a top down manner which is guided

by a cost model. Additionally, there are variants of spatial keyword queries in the literature such as m-closest keyword search [105, 106] which returns spatially closest objects that match with given m keywords, collective keyword search [107] which retrieves group of objects that cover all the given keywords, to name a few.

Location aware publish/subscribe query is another type of spatial keyword queries that is closely related to this thesis. Such a query reports geo-tagged event notification to the relevant subscribers, where the relevance is measured either by boolean matching or similarity based method. Guo et al. [108] propose an efficient location-aware pub/sub system called Elaps that updates the moving subscriber with events within a given spatial range and also match with a given boolean expression. They also introduce an indexing structure called BEQ-tree to support spatial subscription matching. Moreover, the similarity based methods are used to address the top-k publish/subscribe problem where the objects are ranked according to the spatial and textual similarity scores. [109] investigates a problem that takes into account the spatial proximity, the textual relevance and also the object recency where the score of an object decays as the time passes. Hu et al. [110] use prefix filtering and spatial pruning techniques to address a problem where only the events which are within a pre-given similarity threshold are returned to the particular subscriber. In Section 6.2.1, we highlight the limitations of closely related work to the problem that is investigated in Chapter 6.

Chapter 3

Category Aware Multi-Criteria Indoor Route Planning

In this chapter, we present our techniques to answer category aware multi-criteria route planning queries (denoted by CAM) in the indoor space. Although the problem of CAM query is NP-hard, we propose exact solutions for the scenarios when the number of query categories is limited. Also, based on a novel *dominance-based* pruning, we propose an efficient approximation algorithm that generates high-quality results. The research presented in this chapter was published in [111].

3.1 Overview

As stated earlier, people spend a significant amount of their time in indoor spaces often in unfamiliar buildings such as shopping malls, airports, and libraries [41, 42]. Recent advances in indoor positioning technologies [112, 113], cheap wireless networks and availability of geo-tagged data have resulted in huge demand for indoor location-based services such as finding nearby indoor

objects, indoor navigation, and route planning to name a few. Route planning is one of the most popular services among both indoor and outdoor users, which assists them in planning a route satisfying their preferences. Specifically, a user may issue a route planning query by providing a source location and a target location along with her preferences as a set of keywords (e.g., restaurant, salon, supermarket). A route planning query returns an optimal route that starts from the source location, passes through at least one location from each given preference and ends at the target location.

Due to its popularity, the route planning query has been extensively studied in the past few years [33, 34, 35, 36, 37]. However, all these techniques are specifically designed for the outdoor spaces and cannot be efficiently extended for the indoor spaces because they fail to exploit the unique properties specific to indoor venues. For example, indoor graphs have a much higher out-degree as compared to the road networks [1]. Furthermore, the object density is much higher for indoor venues, e.g., the number of POIs (e.g., restaurants, fuel stations) on the vertices of road networks is typically small whereas the number of objects in a single room (e.g., products in a supermarket) of indoor venues may be in thousands. Thus, specialized techniques are required to answer route planning queries in indoor venues.

Inspired by the above, in this chapter, we provide the first set of techniques to answer an important route planning query with various applications in different scenarios. Consider a user who is in the car park of a large shopping centre and has a list of items to buy (e.g., a wine bottle, a bunch of flowers, a cake, and a wristwatch). She may want to find an optimal route such that the total distance she needs to walk and the total price she pays to purchase all these items are minimized. She may use a *category aware multi-criteria route planning* query, denoted as CAM, which takes as input a set of categories (e.g.,

the list of items she wants to purchase) and a scoring function, and returns the route that passes through at least one object of each category and has the minimum score where score of each route is computed using the user-defined scoring function considering the total length of the route and total price of the items along the route.

In contrast to traditional route planning queries that only consider a single criterion (i.e., distance), *category aware multi-criteria route planning* queries could retrieve optimal route considering multiple criteria such as the total length of the route, total price, total rating of items, and total waiting time for the activities etc. Consider another example of a user in an airport who is running late for a flight and needs to withdraw money from an ATM, grab a coffee, and needs to go to a service desk before she checks in. For such a user, the total length of the route is important as well as the total waiting time at the ATM, coffee machine and service desk. Therefore, she may issue a CAM query where the scoring function is used to compute the score of a route considering its total length and the total waiting time at each facility (i.e., ATM, coffee machine and service desk) along the route.

To the best of our knowledge, we are the first to study the route planning queries where the score of a route is computed using not only its total length but also other relevant attributes such as total price and total waiting time etc. Although we show that a CAM query is NP-hard in the number of categories, it can be solved efficiently if the number of categories is small (which is typically the case in many real-world applications). To this end, we propose two efficient exact algorithms to answer CAM queries that cleverly exploit the properties specific to the indoor spaces and outperform the techniques adapted from outdoor techniques by up to three orders of magnitude. However, for a large number of categories, the exact algorithms become increasingly expensive. In

this case, we need approximation solutions that generate high-quality results in a reasonable time. To address this issue, we present an efficient approximation algorithm that utilizes a novel *dominance-based* pruning to significantly reduce the number of possible candidate routes while maintaining high-quality results. Our extensive experimental study shows the superiority of our solutions over the existing outdoor techniques.

3.2 Contributions

We summarize our contributions below.

- We propose the category aware multi-criteria route planning (CAM) query and show that the problem of solving CAM query is NP-hard.
- We present an efficient exact solution called BFNE to answer CAM queries when the number of query categories is limited. Then we introduce an improvement of BFNE called BFNE-opt that utilizes a novel indoor graph traversal method in network expansion.
- We present a fast approximation algorithm to retrieve high-quality results for CAM queries. Then we suggest a pre-processing phase that employs a novel *dominance-based* pruning technique to accelerate the performance of the proposed algorithm.
- We conduct an extensive set of experiments on a large real-world shopping centre containing real products. The experiments demonstrate that our algorithms outperform state-of-the-art techniques in terms of both the runtime and the quality of results.

The rest of this chapter is organized as follows: Section 3.3 presents some preliminaries related to this chapter and formulates the problem, Section 3.4

describes the proposed exact solutions, Section 3.5 presents the proposed approximation solution, Section 3.6 reports the experiment results and Section 3.7 concludes the chapter.

3.3 Preliminaries

In this section, we formally define the problem of category aware multi-criteria route planning query and prove the hardness of the problem in Section 3.3.1. In Section 3.3.2, we briefly discuss the limitations of existing techniques in both outdoor and indoor spaces. Notations used in this chapter are summarized in Table 3.1.

3.3.1 Problem Definition

Definition 3.1 (Indoor Graph) *The indoor space is represented by a D2D graph* $\mathcal{G} = (D, E)$ *where* D *is the set of vertices (i.e., doors) and* E *is the set of edges. For two adjacent vertices* $d_i, d_j \in D$*, we define an edge between them as* $e(d_i, d_j)$ *with weight* $w(d_i, d_j)$ *representing the indoor distance between the doors* d_i *and* d_j*, i.e.,* $dist(d_i, d_j)$*.*

Indoor objects. Let $p_i \in \mathcal{P}_j$ be an indoor point representing an indoor object. Each point p_i is associated with a category $c_j \in \mathcal{C}$ and a static score denoted by $s(p_i)$. For example, a *Samsung Galaxy S5* phone is an indoor object belongs to a category *phones*, whereas its location is a point in the indoor venue and its static score may refer to its price. The static score of a point is the score that does not depend on the query, e.g., the price of an item, waiting time at a service, rating of an item etc.

Definition 3.2 (Route) *A route* $R_{1\rightarrow m} = \langle p_1, \ldots, p_m \rangle$ *denotes a path from indoor point* p_1 *to* p_m *where* $\langle p_i, p_{i+1} \rangle$ *is the shortest path between two points.*

Table 3.1: The summary of notations

Notation	Definition
p_i	An indoor point
c_j	A category
$s(p_i)$	The static score of point p_i
q_i	A CAM query
d_i	A door in indoor space
I_j	An indoor partition
p_s/p_t	The start/end point of a route
ψ	A set of categories
$\mathcal{P}_i$	The set of indoor points of category c_i
R_i^k	k^{th} path from p_s to p_i
L_i^k	k^{th} Label at point p_i
$\hat{R}_{n,j}$	Solitary route at p_n covering c_j
$p_a \prec_k p_b$	The point p_a dominates p_b w.r.t door d_k
$R_a \prec R_b$	The route R_a dominates R_b
Dom_a^k	The dominated set of point p_a w.r.t door d_k

Definition 3.3 (Travel cost) *Given a route $R_{1\to m} = \langle p_1, \ldots, p_m \rangle$, the travel cost of route R is computed as follows,*

$$T(R_{1\to m}) = \sum_{i=1}^{m-1} dist(p_i, p_{i+1}) \tag{3.1}$$

where $dist(p_i, p_{i+1})$ denotes the indoor distance between two points in the given route $R_{1\to m}$.

Definition 3.4 (Static cost) *Given a route $R_{1\to m} = \langle p_1, \ldots, p_m \rangle$, let $R.\psi = \langle c_1, \ldots, c_m \rangle$ be the set of categories covered by R where $|R.\psi| = m$ and p_i denotes an indoor point that covers $c_i \in R.\psi$. Hence, the static cost is computed as follows,*

$$S(R_{1\to m}) = \sum_{i=1}^{m} s(p_i) \tag{3.2}$$

where $s(p_i)$ denotes the static score of the indoor point p_i.

Definition 3.5 (Cost function) *We determine the cost of a route $R_{n\to m}$ in terms of travel cost and static cost, as follows,*

$$Cost(R_{n\to m}) = \alpha \cdot T(R_{n\to m}) + (1-\alpha) \cdot S(R_{n\to m}) \tag{3.3}$$

Here, α is a query parameter (user-defined) that lies between 0 and 1 to control the preference of travel cost over static cost.

Definition 3.6 (Category Aware Multi-criteria route planning (CAM) query) *Given an indoor space, a category aware multi-criteria route planning query $q = \langle p_s, p_t, \psi \rangle$ where p_s, p_t denotes the source point and the target point of the route, and $q.\psi = \langle c_1, \ldots, c_m \rangle$ denotes a set of unique categories that describes the user preferences. A route from the point p_s to the point p_t, that passes through at least one indoor point from each given category, is called a complete candidate route. Moreover, a CAM query returns a route subject to:*

$$R^{opt}_{s\to t} = \underset{R_{s\to t}\in F(q)}{\boldsymbol{arg\ min}}\, Cost(R_{s\to t}) \tag{3.4}$$

where $F(q)$ is the collection of all complete candidate routes for the given query q.

Theorem 3.1 *The problem of solving a CAM query is NP-hard.*

Proof This problem can be reduced from the classical travelling salesman problem (TSP) which is NP-hard. Given a graph in which each edge has a length, let both start and end points equal to a node v_0, each given category is covered by a node v_i with $s(v_i) = 0$ where $i = \{1 \ldots m\}$ and all the other nodes contain non-query categories. Clearly, the problem of solving CAM query is identical to the TSP. Thus, the problem of solving CAM problem is NP-hard.

This proof is similar to the existing proofs of related problems. We have included this proof in the thesis for the sake of completeness.

3.3.2 Limitations of Existing Techniques

The solutions that are presented in [33] cannot be applied to answer CAM queries because they take into account only the distance in finding an optimal route while problem of CAM queries consist of multiple criteria such as distance, static costs. The CAM queries are different from OSR queries [34, 59, 60, 61, 62, 63] since they do not take into account an order of visiting query categories. Therefore, the solutions of these studies are inapplicable in processing CAM queries.

[35, 36, 64] study variants of route planning queries that take into account different constrains. We find these works have different aims compared to CAM queries. Yao *et al.* [37] study a similar problem to CAM. Hence, we employ extensions of their solutions in our experiments to evaluate our proposed solutions. [65] are the first to study the indoor trip planning queries. We observe that the pruning rules which they propose are only based on the distance and become inapplicable when the additional attributes and a user-defined query parameter, i.e., alpha value, appear in the scoring function. Also, they assume that the number of categories in an indoor venue is small (Note that this is not the number of query categories), i.e., less than 10. Therefore, it is computationally prohibitive to employ their pruning rules when the category domain is huge. Thus, their pruning rules and techniques do not hold for CAM settings.

3.4 Exact Solutions

Even though the problem of CAM query is NP-hard, it can be solved efficiently when the number of categories is small. Hence, we propose two exact algorithms to answer CAM queries that cleverly exploit the unique properties specific to the indoor space. We present our first exact solution in Section 3.4.1. Section 3.4.2 explains how to compute the *lower bound cost* for a given candidate route. In Section 3.4.3, we present our second exact solution which utilizes a novel indoor graph traversal method.

3.4.1 BFNE Algorithm

A brute force approach of answering a CAM query is to conduct an exhaustive search starting from the source point by enumerating all possible routes to the target point and return the route with minimum route cost after all possible routes are found. Even though the brute force method guarantees the optimal solution, the exhaustive search is prohibitively expensive in practice. Thus, we propose an approach called Best First Network Expansion (denoted by BFNE) that avoids enumerating unnecessary candidate routes.

The basic idea of the BFNE approach is to generate candidate routes based on the network expansion method while selecting the best candidate route. Hence, we start from the source vertex and keep generating candidate route routes by expanding the most promising vertex. To determine such a vertex, we utilize the concept of the traditional A* search where we compute *lower bound cost* for a given candidate route $R_{s \to n}$ as follows,

$$LB_n(R_{s \to t}) = Cost(R_{s \to n}) + Cost^{est}(R_{n \to t} | R_{s \to n}) \tag{3.5}$$

where $Cost(R_{s\to n})$ denotes the cost of the covered route segment, i.e., the route from the source vertex d_s to the current vertex d_n. $Cost^{est}(R_{n\to t}|R_{s\to n})$ is the heuristic function that estimates the cost of the remaining route segment, i.e., the route from the current vertex d_n to the target vertex d_t. We comprehensively discuss the process of obtaining the lower bound cost in Section 3.4.2.

Label Pruning Technique

Since the indoor graph is modelled based on the doors in the indoor space, graph traversal is basically travelling from door to door (or partition to partition). Hence, we observed that there can be more than one candidate route from d_s to d_t through a door d_i (i.e., a graph vertex). Intuitively, we can prune a large number of candidate routes by expanding only the best candidate route at each vertex. Based on this strategy, we propose a pruning technique that maintains only the best candidate route at each vertex. Before we proceed to explain the pruning technique, we introduce the following definitions.

Definition 3.7 (Route Label) *A label (denoted by L_i^k) is generated in the format of $\langle R_i^k, L_i^k.\psi, LB_i^k(R_{s\to t})\rangle$ where R_i^k denotes the k^{th} route from p_s to p_i, $L_i^k.\psi$ is the already covered categories of R_i^k and $LB_i^k(R_{s\to t})$ is the lower bound cost of R_i^k.*

Definition 3.8 (Label Dominance) *The dominance of a label over another label is decided as follows. We say that label L_i^a **dominates** label L_i^b iff the $L_i^b.\psi \subseteq L_i^a.\psi$ and $LB_i^b(R_{s\to t}) \geq LB_i^a(R_{s\to t})$.*

A label L_i^b is said to be dominated by another label L_i^a, if and only if the label L_i^b covers a subset of the categories covered by the label L_i^a and the route R_i^b have a higher lower bound cost than the route R_i^a. Thus, we prune label L_i^b

since the label L_i^a dominates the label L_i^b. Implicitly, we prune all the unnecessary candidate routes that do not lead to an optimal solution by keeping only the non-dominated labels at each indoor point.

Inverted VIP-tree. In this study, we utilize VIP-tree [1] which is the state-of-the-art indoor index as our indexing structure. In order to support category based filtering, we modify VIP-tree by storing inverted files in each tree node. An inverted file consists of a list of all the unique categories that appear in any indoor partition of that node, and for each category, a list of indoor partitions in which it appears. Additionally, in each tree node, we maintain a summary for minimum static scores of each covered categories.

We provide more details of the usage of these summary lists in Section 3.4.2.

Now we present our first exact solution called BFNE algorithm. Basically, we traverse the D2D graph $\mathcal{G}$ in order to obtain the optimal route. Unlike the outdoor techniques, indoor points are not mapped on to either vertices or edges. We maintain such points in the actual locations inside the corresponding indoor partitions to obtain more realistic results. Hence, we need to carefully handle these points as they lie in the Euclidean space and thousands of points may reside in an indoor partition. For the sake of simplicity, the source/target points are considered as vertices. Note that, all the techniques and definitions that we present can be easily applied for the case where source/target are indoor points.

As Algorithm 1 describes, we use a min-priority queue $\mathcal{Q}$ to organize the labels by the increasing order of the lower bound cost, i.e., $LB_i^k(R_{s\to t})$. We terminate the algorithm when the queue is empty (line 4) or all the labels in the queue have lower bound costs greater than the current optimal route cost

Algorithm 1: BFNE Algorithm

Data: D2D graph $\mathcal{G}$, $q = \langle d_s, d_t, \psi \rangle$, Inverted VIP tree $\mathcal{T}$
Result: The optimal route R^{opt}

Initialize a min-priority queue $\mathcal{Q} \leftarrow \emptyset$;
Create route label $L_s^0 \leftarrow \langle (d_s), \emptyset, 0 \rangle$;
$\mathcal{Q}$.enqueue(L_s^0);
while $\mathcal{Q}$ *is not empty* **do**
    $L_i^k \leftarrow \mathcal{Q}$.dequeue();
    if $LB_i^k(R_{s \to t}) \geq Cost(R^{opt})$ **then**
        break;
    end
    Obtain R_i^k from L_i^k ;
    foreach *edge* (d_i, d_j) **do**
        $P^* \leftarrow$ current indoor partition;
        `// categories in current partition`
        $\Psi \leftarrow P^*.\psi \cap q.\psi \setminus R_i^k.\psi$;
        $\Delta \leftarrow$ all possible combinations of Ψ;
        foreach *category combination* δ *in* Δ **do**
            $R_j^l \leftarrow generateRoute(R_i^k, P^*, d_i, d_j, \delta)$;
            if d_j *is* d_t **then**
                if $q.\psi \setminus R_j^l.\psi = \emptyset$ ***AND*** $Cost(R_j^l) < Cost(R^{opt})$ **then**
                    $R^{opt} \leftarrow R_j^l$;
                end
            else
                `// lower bound cost`
                Compute $LB_j^l(R_{s \to t})$
                if $LB_j^l(R_{s \to t}) < Cost(R^{opt})$ **then**
                    $L_j^l \leftarrow \langle R_j^l, R_j^l.\psi, LB_j^l(R_{s \to t}) \rangle$;
                    $\mathcal{Q}$.enqueue(L_j^l);
                end
            end
        end
    end
end
return R^{opt}

(line 6-7), which implies that none of the existing candidate routes in the queue leads to an optimal solution which is better than the current optimal route. In each iteration, we dequeue the label L_i^k with minimum lower bound cost, from the queue (line 5). Then, the next promising candidate route is obtained from the dequeued label and extended to new candidate routes by expanding all the adjacent edges (line 9-21). As we maintain the indoor points in the Euclidean space, we need to identify the current indoor partition that is being accessed. Since an edge of the indoor graph actually represents an indoor partition, the edge information is exploited to identify the current partition (line 10). Then, we obtain a list of uncovered categories, i.e., $q.\psi \setminus R_i^k.\psi$, covered by the current partition (denoted by Ψ) and generate the candidate routes via indoor points inside the partition covering all possible category combinations (denoted by Δ) (line 12). For example, let $\Psi = \{c_1, c_2\}$. Then, there are following possible combinations, i.e., $\Delta = \{\{\emptyset\}, \{c_1\}, \{c_2\}, \{c_1c_2\}, \}$. We use a *progressive neighbour exploration* [34] based algorithm to find the optimal route inside an indoor partition. Basically, we obtain a route from a given door, i.e., d_i to another door, i.e., d_j, of the particular partition covering the given a set of categories, i.e., δ, by progressively finding the *nearest neighbour* points from the given set of categories. After a candidate route is generated via an indoor partition, we check whether the candidate route has reached the target door (i.e., d_t), it covers all the query categories and the route cost is less than the cost of the current optimal route. If it satisfies all three conditions, then the current candidate route is selected as the current optimal route (line 15-17). Otherwise, we compute the lower bound cost using Algorithm 2 and construct a new route label. Before the label is enqueued, the label dominance is verified where only the dominant labels are enqueued (line 17-22). Finally, the optimal route is returned (line 23).

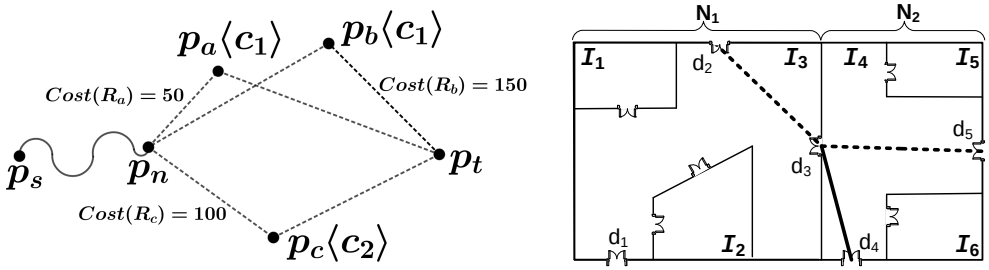


Figure 3.1: Example of (a) solitary routes and (b) leaf node selection

3.4.2 Lower Bound Cost

As we mentioned earlier, we adopt the concept of traditional A^* search to compute the lower bound cost of a given candidate route. In order to obtain an adequate lower bound cost, we take into account both travel and static cost in our heuristic function, i.e., $Cost^{est}(\dots)$. Thus, the estimation of remaining route segment, i.e., $R_{n\to t}$, is conditioned on the covered route segment, i.e., $R_{s\to n}$. Basically, we take into account the uncovered categories of the route segment $R_{s\to n}$ for a tight estimation while maintaining the admissibility of the heuristic function.

Definition 3.9 (Solitary Route) *Let $R_{n\to t}$ be a route starts from a vertex d_n and ends at the target vertex d_t covering a given category c_j. Then, a solitary route is determined subject to:*

$$\hat{R}_{n,j} = \underset{\forall R_{n\to t}}{\boldsymbol{arg\ min}}\ \alpha \cdot T(R_{n\to t}) + (1-\alpha)\cdot S(R_{n\to t}) \tag{3.6}$$

Our approach to estimate the cost for a route $R_{n\to t}$ conditioned on a route $R_{s\to n}$ consists of two phases. First, we obtain a solitary route per uncovered category. For example, Figure 3.1 shows a partial candidate route from the source vertex d_s to the vertex p_n, and the target vertex d_t along with another

three indoor points p_a, p_b and p_c. The category which is covered by an indoor point is mentioned next to each point. The routes that go through each point are R_a, R_b and R_c respectively. Assume that $q = \{d_s, d_t, \{c_1, c_2, c_3\}\}$ and the route from d_s to d_n covers the category c_3. According to the Definition 3.9, R_a and R_c are identified as solitary routes for categories c_1 and c_2 respectively. They can be denoted by $\hat{R}_{n,1}$ and $\hat{R}_{n,2}$.

These solitary routes articulate the individual costs of covering each uncovered category. Thus, selecting the maximum cost among these costs gives us a tight lower bound for cost of the remaining route segment, i.e., $R_{n \to t}$. Hence, we obtain the $Cost^{est}(R_{n \to t} | R_{s \to n})$ as follows,

$$Cost^{est}(R_{n \to t} | R_{s \to n}) = \underset{c_j \in \Psi}{\textbf{arg max}}\, Cost(\hat{R}_{n,j}) \tag{3.7}$$

where $\Psi = q.\psi \setminus R_{s \to n}.\psi$. According to the previous example, the cost of $\hat{R}_{n,2}$ is selected as the estimated cost of $R_{n \to t}$ conditioned on $R_{s \to n}$ since it is the solitary route with the maximum route cost.

Furthermore, we observed that obtaining a solitary route for a given category via an indoor point is computationally very expensive as every point in the indoor space that belongs to the given category needs to be examined. Thus, we obtain solitary routes via indoor partitions since the number of indoor partitions is very small compared to the number of points (i.e., indoor objects) in an indoor venue. Even though this gives a loose lower bound we utilize this approach while only paying a small penalty in query processing. The summary lists that are stored at each tree node come in handy as we need to compute minimum static scores for each partition without accessing indoor points in the partition. In such a list, we store the minimum static scores for each category that appear in the corresponding partition so that the minimum

static cost of a partition can be easily determined.

As Algorithm 2 illustrates, we traverse each level of the inverted VIP-tree starting from the root node (line 3) and compute the costs of the solitary routes via all the sub-components, i.e., tree nodes and partitions, at each level (line 11-16) for the set of categories (denoted by Ψ) that are not covered by the current candidate route (line 2). Such a sub-component is enqueued into a min-priority queue $\mathcal{Q}$ with the cost of the corresponding solitary route as the key value. We utilize the summary list of minimum static scores at such a sub-component to compute the solitary route cost without accessing the indoor objects in the particular sub-component. Hence, the dequeued element from the queue (line 5) can be (i) an intermediate tree node, (ii) a leaf node or (ii) a partition. If the dequeued element is an intermediate tree node, then we enqueue the solitary routes via its child nodes (line 14-16). If the dequeued item is a leaf node, then we enqueue the solitary routes via the partitions belong to the particular leaf node (line 11-13). If the dequeued item is a partition, then we update the uncovered category set by removing the categories covered by the partition (line 7). By doing this, we keep track of the solitary routes per uncovered categories. When all the solitary routes per uncovered categories have been found (line 8), intuitively, the last solitary route give the maximum cost according to the Equation 3.7. Thus, we determine the cost of the last solitary route as the estimated cost, i.e., $Cost^{est}(R_{n\rightarrow t}|R_{s\rightarrow n})$ (line 9-10). Finally, we return the $LB_n(R_{s\rightarrow t})$ according to the Equation 3.5 (line 17-18).

3.4.3 BFNE-Opt Algorithm

As we discussed in the previous section, BFNE travels from door to door and generates candidate routes inside indoor partitions if the corresponding

Algorithm 2: Lower bound Cost Computation

Data: Inverted VIP-Tree $\mathcal{T}$, Route label L_i^k
Result: Lower bound cost $LB^i(R^k_{s\to t})$

Initialize a min-priority queue $\mathcal{Q} \leftarrow \emptyset$;
$\Psi \leftarrow q.\psi - L_i^k.\psi$;
$\mathcal{Q}$.enqueue($\mathcal{T}.root, 0$);
while $\mathcal{Q}$ *is not empty* **do**
    element $\leftarrow \mathcal{Q}$.dequeue();
    if *element is a partition* **then**
        $\Psi \leftarrow \Psi - element.\psi$;
        if $\Psi == \emptyset$ **then**
            $Cost^{est}(R^k_{i\to t}|R^k_{s\to i}) \leftarrow element.key$;
            break;
        end
    end
    else if *element is a leaf node* **then**
        foreach *partition* $\mathcal{I}$ *of element* **do**
            $\mathcal{Q}$.enqueue($\mathcal{I}, Cost^*(\hat{R}_{n,j})$);
        end
    end
    else
        foreach *childNode* $\mathcal{N}$ *of element* **do**
            $\mathcal{Q}$.enqueue($\mathcal{N}, Cost^*(\hat{R}_{n,j})$);
        end
    end
end
$LB^i(R^k_{s\to t}) \leftarrow Cost(R^k_{s\to i}) + Cost^{est}(R^k_{i\to t}|R^k_{s\to i})$;
return $LB^i(R^k_{s\to t})$

partitions contain indoor points belong to the uncovered query categories. Intuitively, the performance of the algorithm can be improved by reducing the number of candidate routes generated in query processing. To tackle this, we introduce a novel indoor graph traversal method called AD2AD graph traversal which visits only a specific type of doors in graph expansion while ignoring the non-optimal candidate routes.

AD2AD Graph. The idea behind the AD2AD graph is similar to D2D graph. Despite, only the *access doors* [1] in the indoor space are taken into account in modeling the AD2AD graph. A door d is called an access door of a vip-tree node N if d connects to the space outside of node N (i.e., one can enter or leave node N via door d). For example, Figure 3.1(b) shows an indoor space consists of 6 indoor partitions and 9 doors. The N_1 and N_2 represent two leaf nodes of the corresponding VIP-tree for this indoor space. Hence, the doors d_1, d_2, d_3, d_4 and d_5 are identified as access doors. The AD2AD graph is constructed by exploiting the leaf level formation of the corresponding VIP-tree. Therefore, an edge is created between two access doors if they are connected to the same leaf node.

Now we present our second exact algorithm (denoted by BFNE-opt) which is an improvement of the BFNE algorithm. In BFNE-opt, we utilize the AD2AD graph traversal that visits only the access doors unless a leaf node consists of indoor partition(s) covering the uncovered query categories. Hence, it avoids generating non optimal routes as early as possible by preventing unnecessary leaf node expansions. Furthermore, when a leaf node consists of partition(s) covering at least one uncovered query category, we generate all possible routes via the particular leaf node by accessing the corresponding partition(s). In the

empirical study, we show the efficiency of AD2AD traversal method compared to the conventional D2D traversal.

However, BFNE-opt is similar to BFNE except for it utilizes the AD2AD graph traversal method for graph expansion. Hence, two modifications occur in Algorithm 1 where we need to determine (i) the *current leaf node* in line 10 and (ii) generate all possible routes through the particular leaf node in line 14. Since we visit from an access door to another access door, we need to determine the leaf node which is currently being accessed (denoted by current leaf node) in order to generate all possible candidate routes through the particular leaf node. Therefore, the current leaf node must be carefully selected to preserve the correctness of the algorithm. Figure 3.1(b) depicts two leaf nodes N_1 and N_2 along with their corresponding access doors d_1, d_2, d_3, d_4 and d_5. Let R_a and R_b be a candidate route where $R_a = \{d_s, \ldots, d_5, d_3\}$ and $R_b = \{d_s, \ldots, d_2, d_3\}$. Assume that $d_3 \rightarrow d_4$ is our current edge (i.e., the solid line) and d_3 is our current door. The dotted lines $d_2 \rightarrow d_3$ and $d_5 \rightarrow d_3$ represent the previous edge of the candidate routes R_a and R_b respectively. We identified three crucial doors (vertices) specific to a given candidate route in determining the current leaf node. Particularly, (i) the current door (e.g., door d_3), (ii) the adjacent door (e.g., door d_4) and (iii) the second last door of the route (e.g., door d_5 corresponds to route R_a). Besides, we also observed two real-world scenarios which is really useful in determining the current leaf node.

Definition 3.10 (Unique/Common leaf node) *Given two doors d_i and d_j along with corresponding leaf nodes N_a, N_b and N_c. Let door d_i belongs to node N_a, N_b and door d_j belongs to node N_b, N_c. Hence, N_a is identified as the **unique** leaf node of d_i w.r.t d_j. And N_b is identified as the **common** leaf node of d_i and d_j.*

One possible scenario is a user enters and exits a leaf node from the same

door after visiting an indoor partition(s) inside the leaf node. The candidate route R_a provides an example for this scenario where user enters and exit the node N_1 from door d_3. We can identify the current leaf node correctly for such a scenario, by checking whether the aforementioned crucial doors (i.e., d_3, d_4 and d_5) belong to the same leaf node and then selecting the *unique* leaf node (i.e., N_1) of the current door (i.e., d_3) with respect to the second last door (i.e., d_5). When those crucial doors do not belong to the same leaf node, it means that user has used different doors to enter and exit the leaf node. The route R_b provides an example for this scenario. Thus, *common* leaf node of the current door (i.e., d_3) and the adjacent door (i.e., d_4) is selected as the current leaf node (i.e., N_2). To obtain all possible candidate routes via a particular leaf node, we use the similar approach used in Algorithm 1 to obtain routes for a partition. In which we progressively find the nearest neighbour partitions covering given categories. i.e., δ, and generate routes via those indoor partitions.

3.5 Approximation Solution

The proposed exact solutions are efficient when the number of query categories is limited. However, for a large number of query categories, such solutions become increasingly expensive that efficient approximation solutions are necessary to answer CAM queries. We present our approximation algorithm in Section 3.5.1, a novel *dominance-based* pruning technique in Section 3.5.2 and an improvement of the proposed approximation algorithm in Section 3.5.3.

3.5.1 GCNN Algorithm

Global Category Nearest Neighbour (GCNN) algorithm is a greedy algorithm that greedily adds an indoor point p to an existing partial candidate route

by minimizing the route cost considering the travel and static costs. Basically, GCNN algorithm starts from the source indoor point p_s and progressively constructs a candidate route by inserting an indoor point covering one of the uncovered query categories. For a given partial candidate route $R = \{p_s, p_1, ..., p_j\}$, the algorithm finds such a point p subjected to:

$$\begin{aligned} Score(p_j, p) = \alpha \cdot (dist(p_s, p) + dist(p_j, p) + dist(p, p_t)) \\ + (1 - \alpha) \cdot s(p) \end{aligned} \tag{3.8}$$

$$cnn(p_j, \mathcal{P}_i) = \mathbf{arg\ min}_{p \in \mathcal{P}_i} Score(p_j, p) \tag{3.9}$$

$$p = \mathbf{arg\ min}_{\forall c_i \in q.\psi \setminus R.\psi} cnn(p_j, \mathcal{P}_i) \tag{3.10}$$

where $cnn(p_j, \mathcal{P}_i)$ returns the *category nearest neighbour* point for a given category c_i with respect to an indoor point p_j. The category nearest neighbour of point p is the closest point to p in terms of both travel and static costs. In order to obtain the category nearest neighbour point covering category c_i for a given point p_j, i.e., $cnn(p_j, \mathcal{P}_i)$, every indoor point p belongs to the particular category c_i, i.e., $p \in \mathcal{P}_i$, is ranked using Equation (3.8). As Equation (3.9) depicts, the point with the minimum ranking score is selected eventually. Then the globally best category nearest neighbour point for the current point p_j is determined using Equation (3.10) and R is updated to $R = \{p_s, p_1, ..., p_j, p\}$. The algorithm terminates when R turns into a complete route where all the query categories are covered. In order to determine such an optimal route, we maintain a min-priority queue where a partial candidate route R is enqueued into the queue by determining the key value equals to $Cost(R) + dist(p_s, p) + dist(p, p_t)$ where p is the recently inserted point. Whenever a candidate route is dequeued from the queue, we find the category

nearest neighbour points for each uncovered category and generate new candidate routes. Then the set of new candidate routes are enqueued into the queue. Intuitively, the candidate route which is dequeued first in next iteration is the answer to Equation (3.10).

Algorithm 3: GCNN Algorithm

```
   Data: A CAM query q = {p_s, p_t, ψ}
   Result: An optimal route R
1  Q ← ∅;
2  R ← {p_s};
3  Q.enqueue(R, 0);
4  while Q is NOT empty do
5  |  R* ← Q.dequeue();                                  // Equation (3.10)
   |  // Let R* = {p_s, ..., p_j}
6  |  Q.clear();
7  |  Ψ ← q.ψ \ R.ψ;
8  |  if Ψ = ∅ then
   |  |  // when route cover all categories
9  |  |  R ← {p_s, ..., p_j, p_t};
10 |  |  break;
11 |  end
12 |  foreach category c_i ∈ Ψ do
13 |  |  p ← cnn(p_j, P_i);                              // Equation (3.9)
14 |  |  R*_i ← {p_s, ..., p_j, p};
15 |  |  key ← Cost(R*_i) + dist(p_s, p) + dist(p, p_t);
16 |  |  Q.enqueue(R*_i, key);
17 |  end
18 end
19 return R
```

As Algorithm 3 illustrates, initially, we enqueue a route $R = \{p_s\}$ with zero as the key value (line 2-3). We terminate the algorithm either when the queue is empty (line 4) or an optimal route is found (line 8-10). In each iteration, we dequeue a candidate route $R^* = \{p_s, ..., p_j\}$ from the queue (line 5) which essentially provides the answer to Equation (3.10) of the previous iteration. After a candidate route is dequeued, we clear the min-priority queue by dequeuing all the routes (line 6). This allows us to maintain the current

optimal partial candidate route in each iteration. Next, the set of categories that is not yet covered, i.e., Ψ, is obtained (line 7). Then for each uncovered category, we get the category nearest neighbour point p using Equation (3.9) and generate a new candidate route by inserting that point into the current candidate route (line 11-13). The key value of a route is determined by taking into account route cost and the distances between the point p and the starting and ending points, i.e., p_s and p_t respectively (line 14). Each route is then enqueued into the queue with its key value (line 15). Finally, the optimal route for the given CAM query is returned (line 16).

For example, Assume that route $R = \{p_s, ..., p_n\}$ where $R.\psi = \{c_3\}$. Let p_a, p_b and p_c be indoor points where p_a, p_c belong to category c_1, i.e., $p_a, p_b \in \mathcal{P}_1$, and p_c belongs to category c_2, i.e., $p_c \in \mathcal{P}_2$. The score of each point with respect to Equation (3.8) is $score(p_a, p_n) = 50$, $score(p_b, p_n) = 90$ and $score(p_c, p_n) = 70$. Assume that $q.\psi = \{c_1, c_2, c_3\}$ and R be the recently dequeued candidate route. Then GCNN algorithm finds the category nearest neighbour point for each uncovered category, i.e., c_1, c_2, using Equation (3.9). Hence, the points p_a and p_c are selected as $cnn(p_n, \mathcal{P}_1)$ and $cnn(p_n, \mathcal{P}_2)$ respectively. Then, new candidate routes $R_1^* = \{p_s, ..., p_{x_j} n, p_a\}$ and $R_2^* = \{p_s, ..., p_n, p_c\}$ are generated accordingly and enqueued into the queue. In the next iteration, R_1^* is dequeued first satisfying Equation (3.10).

Determining an optimal route in GCNN algorithm is very expensive since all the indoor points belong to the uncovered categories are ranked in each iteration to find category nearest neighbours. Thus, the number of times, the ranking operation is executed in obtaining an optimal route is $\mathcal{O}(n \cdot m^2)$, where n is the average number of indoor points per category and m is the total number of query categories. Intuitively, the performance of GCNN algorithm is decreased as m and n are increased. Thus, the performance of the algorithm

can be accelerated by reducing the indoor points accessed by the algorithm in query processing.

3.5.2 Dominance-based Pruning

In this section, we introduce a novel pruning technique called dominance-based pruning which is capable of identifying the indoor points that are highly unlikely to be selected in constructing an optimal route. It takes into account both travel and static costs in determining such points and independent of query preference value, i.e., α. Thus, these indoor points can be safely pruned in advance of the query processing. In the experiments, we show that the ratio of false positives is significantly small in which the deviation of the quality of approximation after the pruning is insignificant. Before we present the pruning technique, we introduce the following definitions.

Definition 3.11 (Point Dominance) *Let p_a and p_b be points belong to category c_i, i.e., $p_a, p_b \in \mathcal{P}_i$, reside in an indoor partition I. Let d_s be one of the doors of I. Then, the indoor point p_a dominates p_b with respect to door d_s, denoted by $p_a \prec_{d_s} p_b$, if and only if $dist(d_s, p_a) < dist(d_s, p_b)$ and $s(p_a) < s(p_b)$.*

Definition 3.12 (Dominated Set) *Let p_a be a point belongs category c_i, i.e., $p_a \in \mathcal{P}_i$. Then dominated set of the point p_a w.r.t door k is defined as follows,*

$$Dom_a^k = \bigcup_{p_a \prec_{d_k} p_j, \forall p_j \in \mathcal{P}_i} p_j \tag{3.11}$$

The dominance of a point over another point can be decided only if both points belong to the same category and reside in the same indoor partition. Let d_s be a door and $p_a, p_b \in \mathcal{P}_i$ be two points in partition I. According to the Definition 3.11, if the point p_a is closer to the door d_s than the point p_b

and also has a static cost less than p_b, then p_a dominates p_b considering the door d_s. Moreover, according to Definition 3.12, the point p_b belongs to the dominated set of p_a considering door d_k.

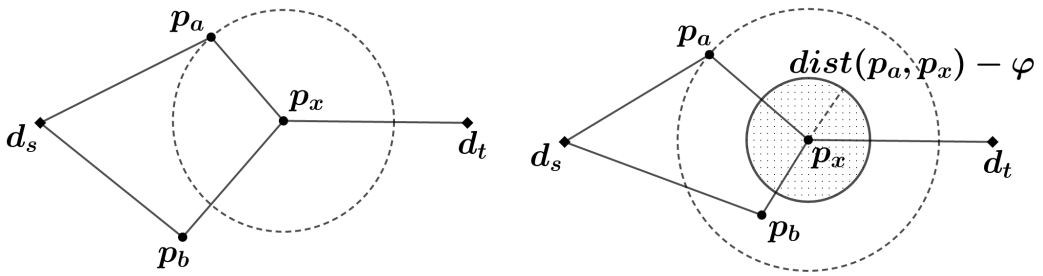
Definition 3.13 (Route Dominance) *Let R_a and R_b be routes inside an indoor partition I, start from door d_s and end at door d_t where $R_a.\psi = R_b.\psi$. Then, R_a dominates R_b (denoted by $R_a \prec R_b$) if and only if $Cost(R_a) < Cost(R_b)$.*

A route can dominate another route only if both routes are inside the same partition, the corresponding starting and ending doors are same, and covering the same set of categories. According to Definition 3.13, if the cost of route R_a is less than the cost of route R_b, then route R_a dominates R_b.

Now we present four important theorems that help to derive our pruning rules. Note that, for all these theorems and pruning rules, we assume that $q.\psi = \{c_m, c_n\}$ and an indoor partition I consist of two doors d_s, d_t. Also, when we say $\mathcal{P}_i$, it means the set of indoor points of the partition I that belongs to category c_i.

Theorem 3.2 *Let routes $R_a = \langle d_s, p_a, p_x, d_t \rangle$ and $R_b = \langle d_s, p_b, p_x, d_t \rangle$ where $p_a, p_b \in \mathcal{P}_m$ and $p_x \in \mathcal{P}_n$. Then, $R_a \prec R_b$ only if $p_a \prec_{d_s} p_b$ and $dist(p_a, p_x) < dist(p_b, p_x)$.*

Proof For the given routes R_a and R_b, if p_a dominates p_b then $dist(d_s, p_a) + s(p_a) < dist(d_s, p_b) + s(p_b)$. Also, we know that, $dist(p_a, p_x) < dist(p_b, p_x)$. By adding both inequalities, $dist(d_s, p_a) + s(p_a) + dist(p_a, p_x) < dist(d_s, p_b) + s(p_b) + dist(p_b, p_x)$. Furthermore, $dist(d_s, p_a) + dist(p_a, p_x) + s(p_a) + dist(p_x, d_t) + s(p_x) < dist(d_s, p_b) + s(p_b) + dist(p_b, p_x) + dist(p_x, d_t) + s(p_x)$. And, $Cost(R_a) < Cost(R_b)$. Hence, $R_a \prec R_b$.



(a) (b)

Figure 3.2: Example of Theorem 3.2 and Theorem 3.3

Theorem 3.3 *Let routes* $R_a = \langle d_s, p_a, p_x, d_t \rangle$ *and* $R_b = \langle d_s, p_b, p_x, d_t \rangle$ *where* $p_a, p_b \in \mathcal{P}_m$ *and* $p_x \in \mathcal{P}_n$, *and* $dist(p_a, p_x) \geq dist(p_b, p_x)$. *Then,* $R_a \prec R_b$ *only if* $p_a \prec_{d_s} p_b$ *and* $dist(p_a, p_x) - \varphi < dist(p_b, p_x)$ *where* $\varphi = dist(d_s, p_b) + s(p_b) - dist(d_s, p_a) - s(p_a)$.

Proof We prove this by contradiction, Assume $R_b \prec R_a$ and $p_b \prec_{d_s} p_a$. Then, $dist(d_s, p_a) + s(p_a) > dist(d_s, p_b) + s(p_b)$. Hence, $0 > \varphi$. Also, we know $dist(p_a, p_x) \geq dist(p_b, p_x)$. By above two inequalities, $dist(p_a, p_x) - \varphi > dist(p_b, p_x)$. Therefore, it must be the case that our assumption is false. So $R_a \prec R_b$ when $p_a \prec_{d_s} p_b$ and $dist(p_a, p_x) - \varphi < dist(p_b, p_x)$.

For given $q.\psi = \{c_m, c_n\}$, the dominance of route $R_a = \langle d_s, p_a, p_x, d_t \rangle$ over $R_b = \langle d_s, p_b, p_x, d_t \rangle$ can be guaranteed if the point p_a dominates p_b and p_a is closer to p_x than p_b (See Figure 3.2(a)). Theorem 3.3 takes into account an instance where the point p_b is closer to p_x than p_a. In this case, $R_a \prec R_b$ can be guaranteed only if the point p_b resides outside the distance threshold $dist(p_a, p_x) - \varphi$ as Figure 3.2(b) illustrates.

For multiple objects. Assume that there is another point $p_j \in \mathcal{P}_m$ within the distance $dist(p_a, p_x)$, where $p_b \prec_{d_s} p_j$. If $dist(p_a, p_x) - \varphi < dist(p_j, p_x)$ where $\varphi = dist(d_s, p_b) + s(p_b) - dist(d_s, p_a) - s(p_a)$, then point p_j can be

ignored since a route via p_j does not dominate R_a. If the indoor points, i.e., p_b, p_j, are visited based on *dominance order*, then distance threshold, i.e., $dist(p_a, p_x) - \varphi$, is guaranteed to be an upper bound as φ is always a lower bound. Visiting the indoor points based on dominance order means that always a point p is visited before visiting a point dominated by p. Moreover, if $dist(p_a, p_x) - \varphi > dist(p_j, p_x)$, then φ needs to be updated with respect to point p_j and checked for $dist(p_a, p_x) - \varphi < dist(p_j, p_x)$. Similarly, all points need to be verified if there is more. Then we can guarantee that R_a dominates R_j where $R_j = \langle d_s, p_j, p_x, d_t \rangle$, $\forall p_j \in \mathcal{P}_m$.

Theorem 3.4 *Let routes* $R_a = \langle d_s, p_a, p_x, d_t \rangle$ *and* $R_b = \langle d_s, p_b, p_y, d_t \rangle$ *where* $p_a, p_b \in \mathcal{P}_m$ *and* $p_x, p_y \in \mathcal{P}_n$. *Then,* $R_a \prec R_b$ *only if* $p_a \prec_{d_s} p_b$, $p_x \prec_{d_t} p_y$ *and* $dist(p_a, p_x) < dist(p_b, p_y)$.

Proof For the given routes R_a and R_b, if p_a dominates p_b and p_x dominates p_y, then $dist(d_s, p_a) + s(p_a) < dist(d_s, p_b) + s(p_b)$ and $dist(d_t, p_x) + s(p_x) < dist(d_t, p_y) + s(p_y)$ respectively. Also, we know that $dist(p_a, p_x) < dist(p_b, p_x)$. By adding them, $dist(d_s, p_a) + s(p_a) + dist(p_a, p_x) + dist(d_t, p_x) + s(p_x) < dist(d_s, p_b) + s(p_b) + dist(p_b, p_y) + dist(d_t, p_y) + s(p_y)$. And, $Cost(R_a) < Cost(R_b)$. Hence, $R_a \prec R_b$.

Theorem 3.5 *Let routes* $R_a = \langle d_s, p_a, p_x, d_t \rangle$ *and* $R_b = \langle d_s, p_b, p_y, d_t \rangle$ *where* $p_a, p_b \in \mathcal{P}_m$ *and* $p_x, p_y \in \mathcal{P}_n$, *and* $dist(p_a, p_x) \geq dist(p_b, p_y)$. *Then,* $R_a \prec R_b$ *only if* $p_a \prec_{d_s} p_b$, $p_x \prec_{d_t} p_y$ *and* $dist(p_a, p_x) - \hat{\varphi} < dist(p_b, p_x)$ *where* $\hat{\varphi} = dist(d_s, p_b) + s(p_b) + dist(p_y, d_t) + s(p_y) - dist(d_s, p_a) - s(p_a) - dist(p_x, d_t) - s(p_x)$.

Proof We prove this by contradiction, Assume $R_b \prec R_a$, $p_b \prec_{d_s} p_a$ and $p_y \prec_{d_s} p_x$. Then, $dist(d_s, p_a) + s(p_a) > dist(d_s, p_b) + s(p_b)$ and $dist(d_t, p_x) + s(p_x) > dist(d_t, p_y) + s(p_y)$. Hence, $0 > \varphi$. Also, we know $dist(p_a, p_x) \geq dist(p_b, p_y)$.

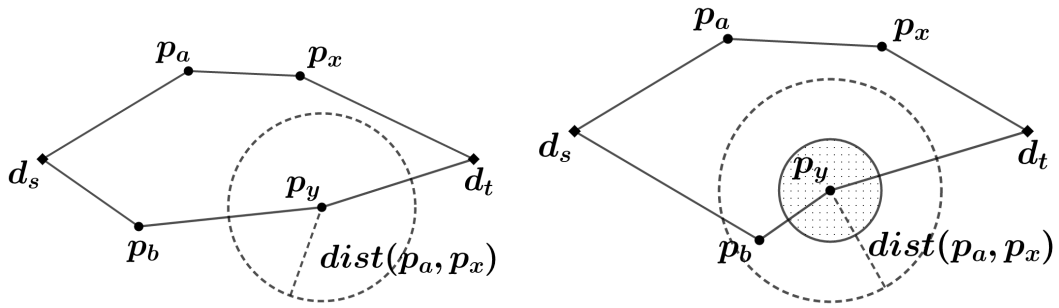


Figure 3.3: Example of Theorem 3.4 and Theorem 3.5

By above two inequalities, $dist(p_a, p_x) - \varphi > dist(p_b, p_y)$. Therefore, it must be the case that our assumption is false. So $R_a \prec R_b$ when $p_a \prec_{d_s} p_b$, $p_x \prec_{d_t} p_y$ and $dist(p_a, p_x) - \hat{\varphi} < dist(p_b, p_x)$.

As Figure 3.3(a) shows, Theorem 3.4 guarantees that a route via points $p_b \in Dom_a^s$ and $p_y \in Dom_x^t$, i.e., $R_b = \langle d_s, p_b, p_y, d_t \rangle$, cannot dominate a route via corresponding points p_a and p_x, i.e., $R_a = \langle d_s, p_a, p_x, d_t \rangle$ when the distance between points p_a and p_x is less than the distance between points in corresponding dominated sets. Theorem 3.5 explains an instance (See Figure 3.3(b)) where $dist(p_a, p_x) \geq dist(p_b, p_y)$. In this case, route R_a dominates route R_b if the distance between the points in dominated sets is greater than the particular distance threshold, i.e., $dist(p_a, p_x) - \hat{\varphi}$.

Next we proceed to introduce our pruning rules which are derived from these theorems. These pruning rules help to filter all the points in an indoor partition that are highly unlikely to be selected in generating an optimal route.

Pruning Rule 1 *Let $p_i \in Dom_a^s$ and $p_b = nn(p_a)$ where $p_a \in \mathcal{P}_m$ and $p_b \in \mathcal{P}_n$. Then, the points p_a, p_b are selected and a point $p_j \in Dom_b^t$ is pruned only if $p_a = nn(p_b)$ and $dist(p_a, p_b) < dist(p_i, p_j)$.*

Proof According to Theorem 3.2, when $p_i \in Dom_a^s$ and $p_b = nn(p_a)$, then $Cost(R_x) < Cost(R_y)$ where $R_x = \langle d_s, p_a, p_b, d_t \rangle$ and $R_y = \langle d_s, p_i, p_b, d_t \rangle$.

Also, when $p_a = nn(p_b)$ and $dist(p_a, p_b) < dist(p_i, p_j)$, Theorem 3.4 guarantees $Cost(R_x) < Cost(R_z)$ where $R_z = \langle d_s, p_i, p_j, d_t \rangle$. Thus, the point $p_j \in Dom_b$ can be pruned since any path covering category c_m and c_n via p_j is dominated by the path R_x.

Pruning Rule 2 *Let $p_i \in Dom_a^s$ and $p_b = nn(p_a)$ where $p_a \in \mathcal{P}_m$ and $p_b \in \mathcal{P}_n$. Then, the points p_a, p_b are selected and a point $p_j \in Dom_b^t$ is pruned when $p_a \neq nn(p_b)$ and $dist(p_a, p_b) < dist(p_i, p_j)$ only if $Cost(R_x) < Cost(R_y)$ where $R_x = \langle d_s, p_a, p_b, d_t \rangle$ and $R_y = \langle d_s, p_i, p_b, d_t \rangle$.*

Pruning Rule 3 *Let $p_i \in Dom_a^s$ and $p_b = nn(p_a)$ where $p_a \in \mathcal{P}_m$ and $p_b \in \mathcal{P}_n$. Then, the points p_a, p_b are selected and a point $p_j \in Dom_b^t$ is pruned when $p_a = nn(p_b)$ and $dist(p_a, p_b) \geq dist(p_i, p_j)$ only if $cost(R_x) < cost(R_y)$ where $R_x = \langle d_s, p_a, p_b, d_t \rangle$ and $R_y = \langle d_s, p_i, p_j, d_t \rangle$.*

Pruning Rule 4 *Let $p_i \in Dom_a^s$ and $p_b = nn(p_a)$ where $p_a \in \mathcal{P}_m$ and $p_b \in \mathcal{P}_n$. Then, the points p_a, p_b are selected and a point $p_j \in Dom_b^t$ is pruned when $p_a \neq nn(p_b)$ and $dist(p_a, p_b) \geq dist(p_i, p_j)$ only if $Cost(R_x) < Cost(R_y)$ where $R_x = \langle d_s, p_a, p_b, d_t \rangle$ and $R_y = \langle d_s, p_i, p_k, d_t \rangle$ for given $p_k \in Dom_b \cup p_b$.*

Note that the Pruning Rule 2, 3 and 4 can be easily proved as the Pruning Rule 1. Intuitively, the pruning rules are capable of identifying the points which are highly unlikely to be selected in generating an optimal route. Hence, such points can be pruned. For example, let I be an indoor partition consists of two doors d_s, d_t and three indoor points $p_a \in \mathcal{P}_m$ and $p_b, p_c \in \mathcal{P}_n$ where $p_b \prec_{d_t} p_c$. Assume that a user who wants to find a route from d_s to d_t covering c_m, c_n categories, visit the point p_a first. Then either the point p_b or p_c needs to be visited before visiting door d_t to get a complete route. Assume that the user visits the point p_b. Then, according to Pruning Rule 1, the

Algorithm 4: *selectPoints* $(\dots)$

Data: Doors d_s, d_t, A pair of categories c_a, c_b
Result: Sets of points $\mathcal{S}_a, \mathcal{S}_b$

while $\mathcal{P}_a \neq \emptyset$ *OR* $\mathcal{P}_b \neq \emptyset$ **do**
  $p_i \leftarrow getPoint\ (d_s, \mathcal{P}_a)$; `// based on dominance order`
  $\mathcal{S}_a \leftarrow \mathcal{S}_a \cup p_i$; `//` p_i `is selected`
  $\mathcal{P}_a \leftarrow \mathcal{P}_a \setminus \mathcal{S}_a$;
  $\overline{\mathcal{P}_b} \leftarrow \mathcal{P}_b$;
  while $\overline{\mathcal{P}_b} \neq \emptyset$ **do**
    $p_j \leftarrow NextNN\ (p_i)$ where $p_j \in \overline{\mathcal{P}_b}$;
    $U_j \leftarrow \forall p_k \in P_a \setminus S_a$ where $dist(p_i, p_j) > dist(p_k, p_j)$;
    if $U_j = \emptyset$ **then**
      $\mathcal{S}_b \leftarrow \mathcal{S}_b \cup p_j$; `//` p_j `is selected`
      $Dom_j^t \leftarrow$ set of points dominated by p_j;
      $\overline{\mathcal{P}_b} \leftarrow \overline{\mathcal{P}_b} \setminus (Dom_j^t \cup p_j)$;
      $\mathcal{P}_b \leftarrow \mathcal{P}_b \setminus prunePonits\ (p_i, p_j, \mathcal{P}_a, Dom_j^t)$
    end
    else
      foreach $p_k \in U_j$ **do**
        `// Ascending order`
        $\varphi = dist(d_s, p_k) + s(p_k) - dist(d_s, p_i) - s(p_i)$;
        $V_k \leftarrow \forall p_m \in U_j$ where $dist(p_m, p_j) < dist(p_i, p_j) - \varphi$;
        if $V_k = \emptyset$ **then**
          $\mathcal{S}_b \leftarrow S_b \cup p_j$; `//` p_j `is selected`
          $Dom_j^t \leftarrow$ set of points dominated by p_j;
          $\overline{P_b} \leftarrow \overline{\mathcal{P}_b} - Dom_j^t$;
          $\mathcal{P}_b \leftarrow \mathcal{P}_b \setminus prunePonits\ (p_i, p_j, \mathcal{P}_a, Dom_j^t)$
        end
        else
          if $p_k \in V_k$ **then**
            `//` p_j `is not selected`
            break
          end
          else
            `// remove points outside the range`
            $U_j \leftarrow U_j \setminus V_k$;
          end
        end
      end
    end
  end
end
return $\mathcal{S}_a, \mathcal{S}_b$

point p_c can be pruned only if $dist(p_a, p_b) < dist(p_a, p_c)$. Because, the route $R_x = \langle d_s, p_a, p_b, d_t \rangle$ dominates $R_y = \langle d_s, p_a, p_c, d_t \rangle$. Hence, the points p_a, p_b are selected as dominant points. Otherwise, when p_c is closer to p_a than p_b. If $dist(p_a, p_c)$ is less than the threshold distance, i.e., $dist(p_a, p_b) - \varphi$, then the points p_a, p_c is selected and the point p_b is pruned.

We proceed to explain Algorithm 4 which utilizes the aforementioned pruning rules to select dominant points of a given indoor partition with respect to a given pair of categories and doors. Clearly, the visiting order of the points is crucial in applying the aforementioned pruning rules. Thus, we visit the indoor points based on the dominance order, i.e., a point p_a is visited before visiting a point $\hat{p_a} \in Dom_a^k$, by utilizing *getPoint* $(d_i, \mathcal{P}_n)$ which returns a point $p \in \mathcal{P}_n$ with the minimum score considering a monotonic ranking function $f(p) = dist(d_i, p) + s(p)$. Initially, the sets of points $\mathcal{P}_a$ and $\mathcal{P}_b$ contain the indoor points belong to category c_a and c_b respectively. If a point is either selected or pruned, then that point is removed from the corresponding point set. Hence, the algorithm is terminated when one of the point sets, i.e., $\mathcal{P}_a$ or $\mathcal{P}_b$, is empty (line 1).

First, we obtain a point p_i belong to category c_a by utilizing *getPoint* $(\cdot)$ (line 2). Then the point p_i is selected as a dominant point of category c_a. We use a temporary point set $\overline{\mathcal{P}_b}$ to maintain the non-pruned set of points per iteration. The inner while loop terminates when $\overline{\mathcal{P}_b}$ is empty, indicating that all the dominant points belong to category c_b based on p_i is selected while the rest of the points is pruned (line 6). After point p_i is selected we find the closest point to p_i, i.e., the point p_j, that belongs to category c_b. Then we check whether there are any points closer to p_j than p_i that belong to category c_a. If not we select p_j and prune all the points dominated by p_j according to the Pruning Rule 1 and the Pruning Rule 3 (line 9-13). Else,

Algorithm 5: *prunePoints* (. . .)

Data: Points p_i, p_j, Sets of points $\mathcal{P}_a, Dom_j^k \subseteq \mathcal{P}_b$
Result: A set of points S_b

1 **foreach** $p_k \in Dom_j^k$ **do**
  `// According to the dominance order`
2 $p_m \leftarrow NN(p_k)$ where $p_m \in \mathcal{P}_a$;
3 **if** $dist(p_i, p_j) < dist(p_k, p_m)$ **then**
4 $S_b \leftarrow S_b \cup p_k$; `// Theorem 3.4`
5 **else**
6 $\hat{\varphi} = dist(d_s, p_m) + s(p_m) + dist(p_k, d_t) + s(p_k) - dist(d_s, p_i) - s(p_i) - dist(p_j, d_t) - s(p_j)$;
7 **if** $d(p_k, p_m) > dist(p_i, p_j) - \hat{\varphi}$ **then**
8 $S_b \leftarrow S_b \cup p_k$; `// Theorem 3.5`
9 **end**
10 **end**
11 **end**
12 **return** S_b

each point that is closer to p_j than p_i is verified and pruned according to the Pruning Rule 2 and the Pruning Rule 4 (line 15-27). Note that, each time we update the $\overline{\mathcal{P}_b}$ by removing the dominated points. Because, if a point is dominated then they cannot be selected in the same iteration. But, we update the $\mathcal{P}_b$ set after verifying that a point can be pruned permanently by utilizing Algorithm 5. While we iterate the set of points closer to the point p_j, if one of the points is within the distance threshold, then the point p_j is not selected as another point in the category c_a creates a better route with the point p_j (line 24-25). Also, we can remove all the points outside the current distance threshold since it provides an upper bound as we explained earlier (line 27). Finally, it returns the sets of selected dominant points per given category (line 28). Algorithm 5 identifies the indoor points that can be pruned based on Theorem 3.4 and 3.5. For a given point p_j, it iterates through each point $p_k \in Dom_j^k$ (line 1) according to the dominance order and gets the nearest neighbours (line 2) to check whether the distance between the points in dominated set is less than the distance between p_i and p_j (line 3). Then

the points are added to the pruned point set (i.e., set of non-dominant points) according to Theorem 3.4 (line 3-4) and Theorem 3.5 (line 5-8). Finally, the set of non-dominant points is returned (line 9).

Algorithm 6: Dominance-based Pruning Algorithm

```
   Data: An indoor partition I, A set of categories Ψ,
   Result: Update set of points in given partition I
1  P_i ← ∅ , ∀c_i ∈ Ψ;
2  foreach {d_i, d_j} ∈ I(N) do
3  |   foreach {c_a, c_b} ∈ Ψ do
4  |   |   S_a, S_b ← selectPoints(d_i, d_j, c_a, c_b) P_a ← P_a ∪ S_a;
5  |   |   P_b ← P_a ∪ S_b;
6  |   end
7  end
   // Update set of points in indoor partition I
8  P_i = P_i , ∀c_i ∈ Ψ;
```

Next we present the dominance-based pruning algorithm that reduces the number of points in an indoor partition by eliminating non-dominant points belong to a given set of categories. As Algorithm 6 illustrates, for a given set of categories Ψ, the dominance-based pruning algorithm determines the set of dominant points for each given category using Algorithm 4 (line 4) and updates the set of points of indoor partition I accordingly (line 7). Since our pruning techniques are based on pairs of doors and categories, we consider all possible combinations of doors and given categories to preserve the correctness of the algorithm. For example, let I be an indoor partition consists of two doors d_{10}, d_{20} where sets of indoor points of I for given categories c_1, c_2 are as follows : $c_1 = \{p_1, p_2, p_3, p_4, p_5\}$ and $c_2 = \{p_6, p_7, p_8, p_9\}$. The outer for loop of the algorithm runs four times corresponding to the number of door pairs, i.e., $\{d_{10}, d_{10}\}, \{d_{10}, d_{20}\}, \{d_{20}, d_{10}\}, \{d_{20}, d_{20}\}$. And the inner for loop runs only once since there is only two given categories. Therefore, Algorithm 4 is executed four times and the selected points are returned in each iteration. Assume that we obtain the results as follows: $\{\{p_1, p_2\}, \{p_8\}\}$,

$\{\{p_2, p_3\}, \{p_7, p_8\}\}$, $\{\{p_1\}, \{p_8\}\}$ and $\{\{p_2\}, \{p_7\}\}$. Then, the set of points of indoor partition I is updated as follows : $c_1 = \{p_1, p_2, p_3\}$ and $c_2 = \{p_7, p_8\}$. The indoor points p_4, p_5 and p_6, p_9 are eliminated from c_1 and c_2 respectively.

3.5.3 GCNN-dom Algorithm

Since the dominance-based pruning is not affected by the query parameters, we introduce a pre-processing phase. We utilize the results of the pre-processing phase to accelerate the performance of the GCNN algorithm. Accordingly, we present an improvement of GCNN algorithm called GCNN-dom. In pre-processing phase, we utilize our dominance-based pruning technique to prune a significant amount of non-dominant indoor points. As Algorithm 6 shows, we iterate the collection of indoor partitions covering a set of *selected categories* while pruning all the non-dominant points. Then we update the inverted VIP-tree corresponding to the remaining dominant points of each partition. The inverted files and the minimum static score summaries of the tree nodes are updated appropriately.

Note that, the effectiveness of the pre-processing approach relies upon the set of selected categories. Because if the set of selected categories has a large number of points in the indoor space, then the dominance-based pruning technique prunes more non-dominant indoor points. We observed that the categories have a large number of points (objects) in an indoor space are the ones that have a higher demand. Intuitively, such categories are more prone to appear in the majority of queries. Hence, we select the categories that are more frequent in real-world queries for the pre-processing. By utilizing the results of pre-processing, GCNN-dom algorithm achieves a significant increment in performance while only paying a small penalty in the quality of approximation.

3.6 Experiments

3.6.1 Experimental Settings

Indoor Venue and Category Datasets. We use Chadstone Shopping Centre [1] as our indoor venue. The Chadstone Shopping Centre which is the largest shopping centre in Australia currently features more than 300 retail outlets across 4 levels, with a total retail floor area over 200,000 m^2. We manually converted the floor plans of the Chadstone Shopping Centre into machine-readable indoor venues. The OpenStreetMap [2] was used to obtain the coordinates of the buildings in which the sizes of indoor partitions (e.g., rooms, hallways) are determined. Moreover, a three-dimensional coordinate system is used to represent an indoor entity in the dataset, in which the first two dimensions represent x and y coordinates of the indoor entry while the third represents the floor number. The corresponding D2D graph consists of 339 vertices and 3867 edges.

We crawled data from the websites of major supermarkets (e.g., Coles, Woolworths, *etc.*) as well as major retail stores (e.g., JB Hi-fi, Big W, *etc.*) and obtained 140,000 objects along with their categories such as dairy, pantry, *etc.* Then, each object was mapped into the corresponding indoor partition (e.g., a retail store) by randomly determining the location of the object inside the partition. Moreover, we replicated the real-world dataset four times and obtained a larger dataset (denoted by REP). The objects are replicated and randomly relocated without changing the associated category.

Query Generation. We generated 5 query sets per dataset to study the

[1] https://www.chadstone.com.au/
[2] https://www.openstreetmap.org/

performance of the algorithms. In order to generate query sets, we took into account a property called the *objects per category* (denoted by Ω) which is the number of objects in the indoor space that belongs to the particular category. First, we identified five category sets with respect to this property, namely XS, S, M, L and XL. The category set XS was obtained by selecting the categories that have 80 - 120 objects in the indoor space. Similarly, the rest of the category sets were obtained from 450 - 550, 950-1050, 1450-1550 and 1950 - 2050 objects respectively. A CAM query is generated by randomly selecting categories from the corresponding category sets while randomly determining the source and target points in the indoor space. 50 queries were generated for each category set accordingly. Moreover, we followed the same procedure to obtain different query sets for the REP dataset.

Competitors. We compare our solutions with the state-of-the-art solutions [37] for outdoor space that solves a similar problem. We extended those solutions to answer the CAM queries by including additional attributes in the route search. And also, we utilized the VIP-tree [1] along with an inverted file in those extensions to support both efficient indoor distance computation and category-based filtering. We compare our solutions with their exact solution called path expansion and refinement (denoted by iPER) and the approximation solution called global minimum path (denoted by iGMP).

Setup. In the real-world dataset, we observed that the categories which have a large number of objects in the indoor space, e.g., the categories in the category set XL, are more clustered while the categories in the category set XS, S, M are well distributed in the indoor space. Hence, as we explained earlier, we replicated the real-world dataset and obtained the REP dataset. And, we

Table 3.2: The parameters used for experiments

Parameter	**Default**	**Range**
Objects per category (Ω)	M	XS, S, M, L, XL
Query categories ($q.\psi$)	6	2, 4, 6, 8, 10
Preference parameter (α)	0.5	0.1, 0.3, 0.5, 0.7, 0.9
Percentage (Δ)	50	100, 80, 50, 20, 10, 0

take into account the same object ranges, i.e., 80- 120, 450 - 550, 950-1050, 1450-1550 and 1950 - 2050, when we were selecting categories for the category sets. This allowed us to select categories per category set that is well distributed in the indoor space. Thus, the query categories of the REP dataset are well distributed and have a higher object density compared to the query categories of the real-world dataset. Table 3.2 shows the default settings we used in our experiments. The percentage (Δ) denotes the percentage of query categories pre-processed. Suppose an approximate method X returns a route R for CAM query where the optimal route is R^{opt} for the same instance. Then, X's approximation ratio $r = cost(R)/cost(R^{opt})$.

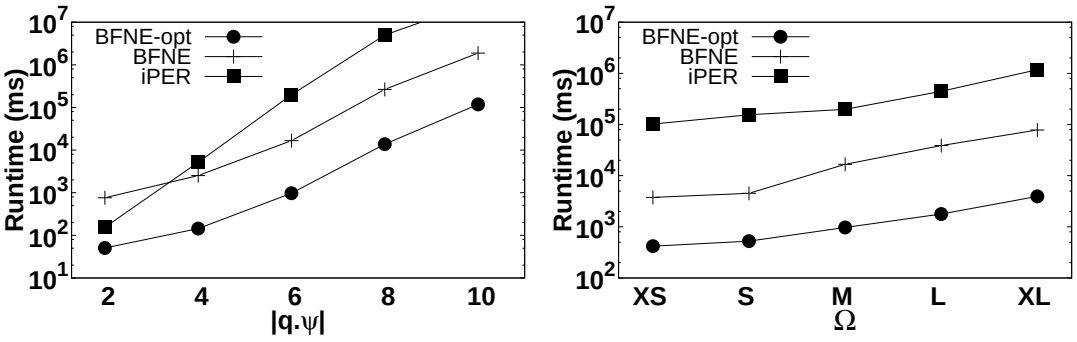
Moreover, all algorithms were implemented in C++ and our experiments were conducted on a Linux platform running on an Intel Core i5 @ 3.30GHz and 4GB RAM.

3.6.2 Experimental Results

In all experiments, we use the default settings while varying a single parameter at a time. Moreover, we report the average runtime in milliseconds and the approximation ratio for each experiment.

Query Performance

The objective of this set of experiments is to investigate the performance of the proposed algorithms on the real-world dataset. Under the default settings,



(a) (b)

Figure 3.4: Runtime of exact solutions on the real-world dataset

each query consists of 6 categories in which each category has around 1000 related objects. Even though there are only 6000 related objects in the indoor space, each algorithm deals with 140,000 objects in query processing. However, all the algorithms (including iPER and iGMP) perform reasonably well since they utilize efficient indoor distance computation and category-based filtering techniques.

Exact Solutions. First, we compare our exact solutions BFNE and BFNE-opt with our competitor iPER. Figure 3.4(a) reports the runtime of exact solutions while varying the number of query categories. Clearly, the performance of all algorithms decreases as $|q.\psi|$ is increased. However, BFNE-opt outperforms BFNE by an order of magnitude when $|q.\psi| = 10$. The reason is, BFNE-opt generates only a less number of candidate routes compared to BFNE since it utilizes AD2AD traversal method for network expansion. Clearly, BFNE-opt is two orders of magnitude better than iPER when $|q.\psi| >= 6$. This is because iPER generates all possible routes to find the optimal route and also generating a route involves expensive ranking operations. For $|q.\psi| = 10$, iPER is outperformed by more than three orders of magnitude.

Figure 3.4(b) shows the runtime of exact solutions under different Ω values.

Since each query consists of 6 categories under the default settings, the average number of related objects in the indoor space is 0.6K, 3K, 6K, 9K and 12K respectively. The runtime of all the algorithms increases as expected. We can see that BFNE and iPER both become much worse as Ω is increased while BFNE-opt maintains a reasonable runtime under all configurations. BFNE-opt is an order of magnitude better than BFNE and three orders of magnitude better than iPER when $\Omega = XL$.

According to the results, BFNE-opt is the most efficient and robust exact solution overall, with practically low runtime. Thus, we compare only BFNE-opt with our approximation algorithms in rest of the experiments.

Approximation Solutions. Figure 3.5(a) and Figure 3.5(c) report the runtime of both exact and approximation solutions on the real-world datasets while varying $|q.\psi|$ and Ω. As Figure 3.5(a) reports, the runtime of the exact solution increases drastically when $|q.\psi|$ is increased. This is because the number of possible combinations to form candidate paths increases with respect to $|q.\psi|$. When the number of query categories is small, i.e., $|q.\psi| \leq 6$, BFNE-opt performs reasonably well where it answers a CAM query in less than 1 second. Clearly, the approximation solutions perform much better under all the settings as they outperform BFNE-opt by three orders of magnitude when $|q.\psi| = 10$. The runtime of GCNN-dom is generally 5-6 times better than GCNN. This is because GCNN-dom ranks a less number of objects as it pre-processes the dataset and eliminates a large amount of non-dominant objects. Overall, the approximation solutions achieve very small runtime to answer CAM query where they take less than 0.2 seconds to complete. GCNN-dom takes 0.09 second while iGMP takes 0.19 seconds. Although GCNN-dom ranks indoor

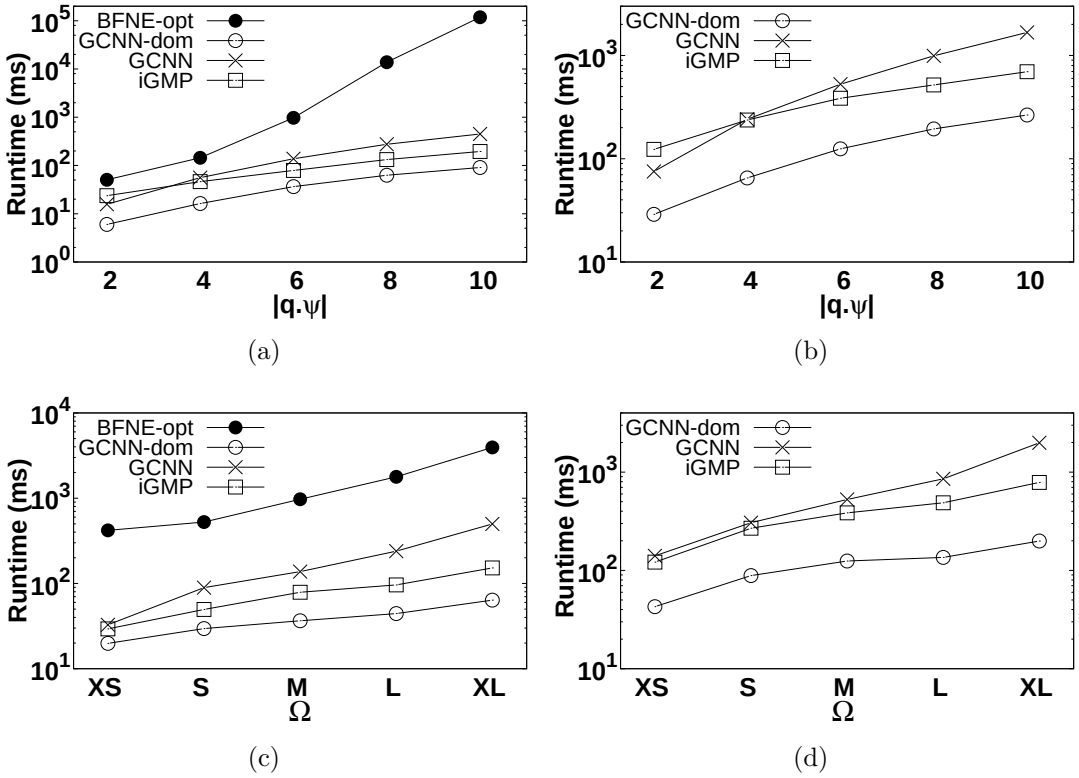


Figure 3.5: Runtime of both exact and approximation solutions on real-world and REP datasets

objects multiple times, it is reasonably efficient compared to iGMP since it utilizes the inverted VIP-tree which supports simultaneous travel and static costs based filtering.

Next, we investigate the performance of our solutions while varying Ω, i.e., the number of indoor objects per query category. As Figure 3.5(c) shows, obviously, the runtime of all solutions increases. As expected, the approximation solutions achieve much better performance as they generate only a significantly small amount of possible candidate routes. Hence, they answer a CAM query in less than 0.1 seconds even for large Ω values. The approximation solutions outperform our exact solution by an order of magnitude when $\Omega = XL$. But, our exact solution performs reasonably well by answering a CAM query less than 5 seconds for the large Ω value. GCNN becomes much worse as Ω in-

creases. The reason is, GCNN has to carry out more ranking operations when the number of related objects in the indoor space increases. Distinctly, GCNN-dom is superior to GCNN as it uses the dominance-based pruning technique in the pre-processing. The results conclude that the dominance-based pruning technique is much effective in accelerating the performance of the proposed approximation solution.

Figure 3.5(b) and Figure 3.5(d) investigate the performance of the approximation solutions on the *REP* dataset. Since the exact solution, i.e., BFNE, failed to finish after a reasonable time for some settings, we have eliminated the results of BFNE and show only the results of the approximation solutions. In the *REP* dataset, the query categories are well distributed and the number of indoor partitions that cover the query categories is very large compared to the real-world dataset. Hence, the runtime of the approximation solutions is increased since the ranking operations become expensive. As Figure 3.5(b) shows, the runtime of GCNN-dom is 0.2 seconds while iGMP is 4 times slower when $|q.\psi| = 10$. As we explained earlier, the number of query categories has a significant impact on the runtime of the algorithms. According to Figure 3.5(d), distinctly, GCNN-dom outperforms iGMP as it takes only 0.2 seconds to answer a CAM query while iGMP takes 0.8 seconds when Ω is very large.

Accuracy of Approximations

This set of experiments is to verify the accuracy of the approximation algorithms. Note that, the results for some of the settings are unavailable since the exact algorithms failed to finish after a reasonable time. Figure 3.6 reports the approximation ratios of algorithms for the experiment in Figure 3.5 where we vary $|q.\psi|$ and Ω. As Figure 3.6(a) reports, clearly, iGMP has the worse

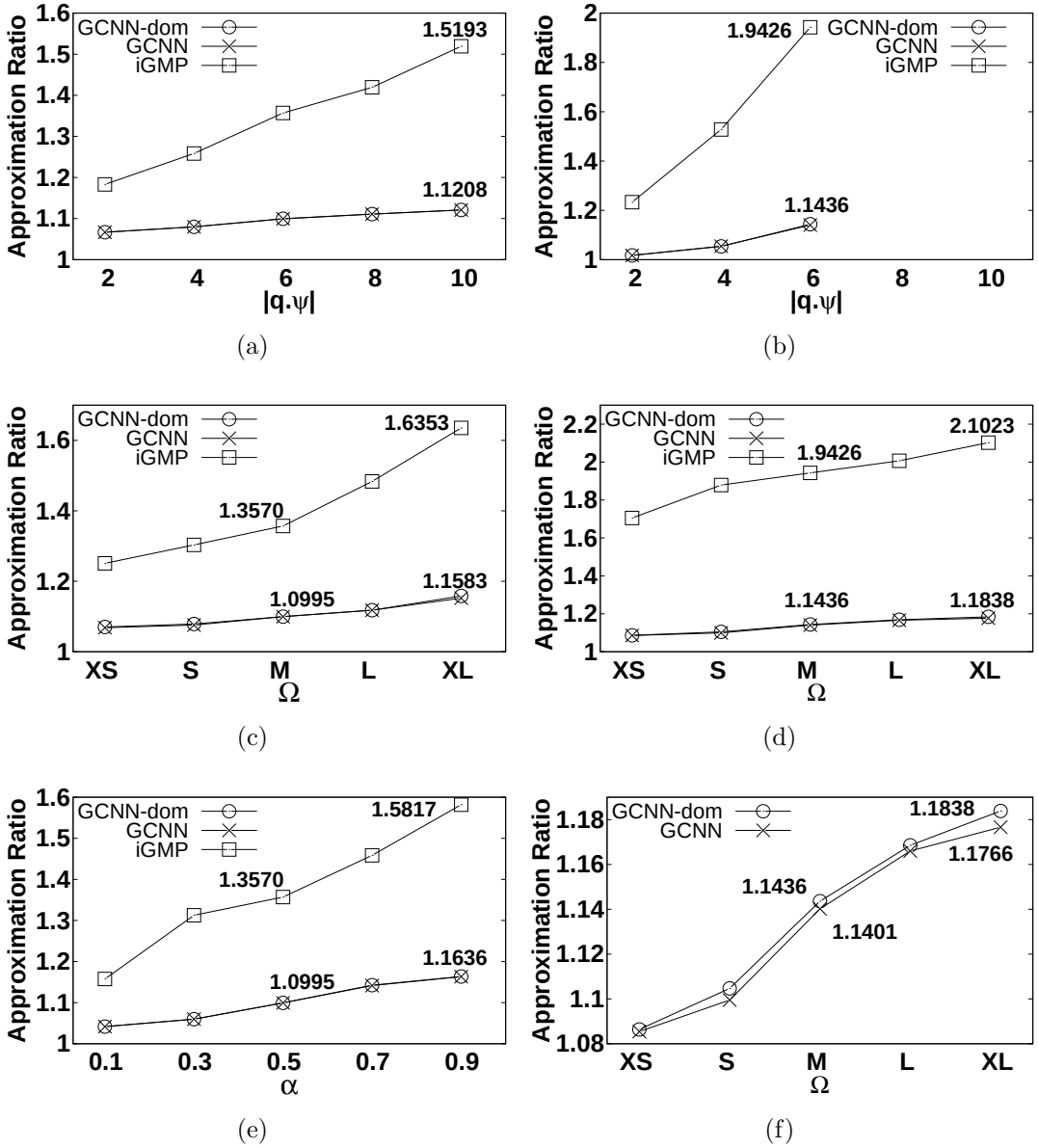


Figure 3.6: Approximation quality of GCNN, GCNN-dom and iGMP

approximation quality under all the settings. When $|q.\psi| = 10$, the approximation ratio of iGMP is slightly higher than 1.5 while both GCNN-dom and GCNN are around 1.1. Evidently, the approximation ratio of the proposed algorithms, i.e., GCNN and GCNN-dom, are almost similar in all cases. This concludes that the dominance-based pruning technique is highly accurate.

According to the Figure 3.6(c), the approximation ratios of all algorithms increase when we increase the Ω. We can see that the approximation ratio of iGMP increases drastically after Ω =M. This is because of the category distribution of the dataset. The approximation ratio of iGMP is 1.6 when Ω =XL, while our algorithms stay close 1.1 in all cases. Figure 3.6(e) shows the approximation ratios of the algorithms when we vary the query preference parameter. The approximation ratios of all algorithms affected by the alpha value. All algorithms have the worse approximation ratio when $\alpha = 0.9$. However, the approximation ratio of our algorithms are under 1.2 while iGMP is close to 1.6. Note that, the approximation ratio of GCNN-dom and GCNN are almost same. Figure 3.6(f) shows the difference between the approximation ratio of our algorithms. The approximation ratio of GCNN-dom has deviated from GCNN by 0.01 when $\Omega = 2000$. For all other cases, clearly, the difference is negligible. This insignificant difference in ratios indicates the precision of dominance-based pruning technique in identifying the incompetent indoor points in the indoor space.

Next, we examine the accuracy of approximations on the *REP* dataset. Figure 3.6(b) shows that the approximation ratio of iGMP is worse in all cases where it is closer to 2 even when $|q.\psi| = 6$. But, GCNN-dom and GCNN stays under 1.2. It clearly shows that the approximation ratio of iGMP drastically increases when the distribution of the categories changes. As Figure 3.6(d) depicts the ratio of iGMP exceeds 2 for the large Ω values. Our algorithms

show much better approximation quality by being consistent around 1.1 under all the settings.

Effect of pre-processing

Figure 3.7 reports the runtime of GCNN-dom and the corresponding pre-processing time while varying the percentage of query categories that have been pre-processed (i.e., Δ). Note that, we denote the pre-processing time by *Pre-proc. Time* in Figure 3.7. The percentage of query categories $\Delta = 50$ denotes that half of the query categories are identified as highly frequent categories for pre-processing while $\Delta = 0$ indicates that no pre-processing is done and $\Delta = 100$ indicates that all the query categories are pre-processed. As Figure 3.7 shows the runtime of GCNN-dom decreases as we pre-process more query categories. The reason is, for large Δ values, GCNN-dom generate only a small number of candidate routes as most of the non-dominant objects have been eliminated in the pre-processing phase. When $\Delta = 100$, GCNN-dom can answer a CAM query in 0.04 milliseconds in which it outperforms iGMP by an order of magnitude. The approximation quality increases as we decrease Δ. But this deviation is insignificant. For example, the difference between the approximation ratio of $\Delta = 0$ and $\Delta = 100$ is 0.001. Hence, we do not report the approximation ratios corresponding to the experiment in Figure 3.7. The results conclude that our dominance-based pruning technique effective in accelerating the performance of the proposed algorithm while only paying a small penalty in quality of approximation.

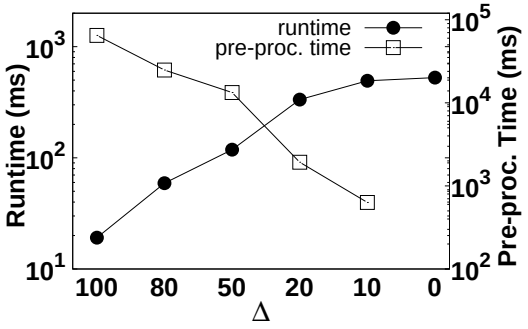


Figure 3.7: Effect of pre-processing

3.7 Conclusions

In this chapter, we define the problem of category aware multi-criteria route planning query, denoted by CAM, which returns a route from a given source indoor point to a target indoor point that passes through at least one indoor point from each given category while minimizing the route cost in terms of travel and static costs. We show that the problem of answering a CAM query is NP-hard in the number of query categories. We propose two efficient exact solutions, namely BFNE and BFNE-opt to answer CAM queries when the number of query categories is limited. However, the exact algorithms become very expensive when the number of query categories is large. Hence, we devise an efficient approximation algorithm called GCNN based on a novel dominance-based pruning technique. The empirical studies on a large real-world dataset demonstrate that the proposed algorithms are highly efficient and offer high-quality results.

Chapter 4

Keyword-aware Skyline Routes Search in Indoor Venues

In this chapter, we study an interesting route planning problem called keyword-aware skyline routes (KSR) query which returns a set of *non-dominated* routes, i.e., a set of skyline routes, based on two attributes, route distance and the number of shops/stores visited. We propose efficient techniques to handle KSR queries in indoor venues. This chapter is based on our research reported in [114].

4.1 Overview

There is a huge demand for indoor location-based services such as finding nearby indoor objects, indoor navigation and route planning to name a few. Route planning is one of the popular services among them. As we mentioned in Chapter 3, the objective of route planning is to assist users in planning a route satisfying their preferences. Specifically, a user may issue a route planning query by providing a source location and a target location along with

her preferences as a set of keywords (e.g., restaurant, salon, supermarket). A route planning query returns an optimal route that starts from the source location, passes through at least one location from each given preference and ends at the target location. The existing route planning queries focus on minimizing the total length of the route that visits all keywords. However, in many real-world applications, especially in indoor venues, users may prefer a slightly longer route that requires visiting fewer stores. Consider a user who is in a large shopping center, wants to buy a few items (e.g., a milk bottle, a loaf of bread and a bunch of flowers). She may interest in finding the optimal route so that she can purchase these items in less time. Therefore, she can issue a traditional route planning query to obtain such an optimal route minimizing the distance/time that she needs to travel. The resulting route of such a query is optimal with respect to the travel distance/time, but it may pass through three different shops/stores to cover the given keywords. Then, she may spend more time waiting at the counter in each store than traveling a little bit further to a shop/store with all the required items. In real-world applications, many users may prefer the route that requires fewer stores (due to various reasons such as waiting times at counters, inconvenience related to visiting a few different stores, etc.). Thus, a route can dominate another route if its distance, as well as the number of stores visited, is smaller. Motivated by this, we study a new interesting route searching problem called keyword-aware skyline routes (KSR) query that returns a set of *non-dominated* routes considering the route distance and the number of shops/stores visited.

Figure 4.1 shows an indoor space consists of 16 partitions and 20 doors. An indoor object inside a partition is illustrated by a solid blue circle and the set of keywords covered by the particular object is mentioned within the curly brackets. Let $q = \{s, t, \{t_1, t_2, t_3\}\}$ be a KSR query and, $R_1 = \{s \rightarrow o_4 \rightarrow$

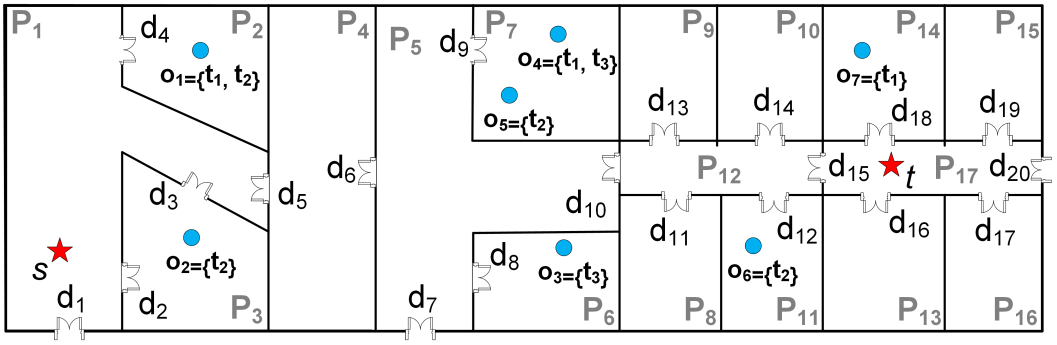


Figure 4.1: Example of an indoor venue

$o_5 \rightarrow t\}$, $R_2 = \{s \rightarrow o_1 \rightarrow o_3 \rightarrow t\}$ and $R_3 = \{s \rightarrow o_2 \rightarrow o_3 \rightarrow o_6 \rightarrow t\}$ be three complete routes. Note that, R_1 covers all query keywords by visiting only one partition, i.e., P_7 while R_2 visits two partitions, i.e., $\{P_2, P_6\}$, and R_3 visits three partitions, i.e., $\{P_3, P_6, P_{11}\}$, respectively. Assume that the routes R_1, R_2 and R_3 are the non-dominated routes, i.e., skyline routes, with route distance 700, 400 and 100 respectively. Hence, any other route in the indoor space cannot dominate these routes in terms of the above mentioned dimensions. For example, let $R_4 = \{s \rightarrow o_2 \rightarrow o_4 \rightarrow t\}$ a complete route in indoor space with route length 600. Then, R_4 is dominated by R_2. These routes are illustrated in Figure 4.2 where the y-axis is the route distance and the x-axis is the number of partitions (i.e., shops/stores) visited. To the best of our knowledge, we are the first to study keyword-aware skyline routes search in the indoor space. In this chapter, we propose techniques for indoor skyline routes query where two attributes are considered in identifying the dominance of a route over another route. Although we show that a KSR query is NP-hard, it can be efficiently solved when the number of query keywords is small, which is typically the case in real-world applications. To this end, we propose an exact algorithm to answer KSR queries that cleverly exploit the properties specific to the indoor space to improve the performance.

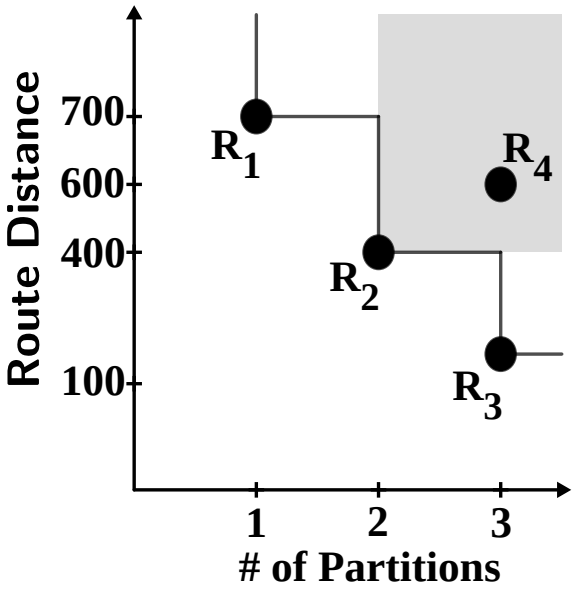


Figure 4.2: Example of skyline routes

4.2 Contributions

We summarize our contributions below.

- We propose the keyword-aware skyline routes (KSR) query and show that problem of answering KSR queries is NP-Hard.
- We present an efficient exact algorithm called GMD to answer KSR queries in real-world applications where the number of query keywords is typically small.
- We conduct an extensive set of experiments on a real-world shopping center containing real products. The experiments demonstrate that our algorithm outperforms a network expansion based solution by two orders of magnitude.

The outline of the chapter is as follows: Section 4.3 formulates the problem and presents the limitations of closely related work, Section 4.4 describes the

proposed techniques, Section 4.5 consists of the experimental evaluation and Section 4.6 concludes this chapter.

4.3 Preliminaries

In this section, first, we introduce some preliminary definitions. Then, we formally define the problem of keyword-aware skyline routes (KSR) query and prove the hardness of the problem. Finally, we present the limitation of existing techniques in both outdoor and indoor spaces.

4.3.1 Problem Definition

Definition 4.1 (Indoor Objects) *Let $p_i \in \mathcal{P}$ be an indoor point representing an indoor object. Each point p_i is associated with a set of keywords denoted by $p_i.\psi$.*

Definition 4.2 (Route) *Given an indoor space, a route $R = \langle p_s, \ldots, p_m \rangle$ denotes a path from indoor point p_s to p_m where $\langle p_i, p_{i+1} \rangle$ is the shortest path between two points. And each point $p_i \in R$ covers one or more query keywords.*

Definition 4.3 (Complete Route) *For a given keyword set, a route from the point p_s to the point p_t, that passes through at least one indoor point from each given keyword, is called a compete route.*

Definition 4.4 (Route Distance) *Given a route $R = \langle p_1, \ldots, p_m \rangle$, the distance of route R is computed as follows,*

$$D(R) = \sum_{i=1}^{m-1} dist(p_i, p_{i+1}) \tag{4.1}$$

where $dist(p_i, p_{i+1})$ denotes the indoor distance between two points in route R.

Definition 4.5 (Partition Count) *Given a route $R = \langle p_1, \ldots, p_m \rangle$, the partition count (denoted by $C(R)$) is the number of unique partitions visited by the route R in order to cover query keywords. $C(R)$ is computed as follows,*

$$N = \bigcup_{i=1, p_i \in P_j}^{m} P_j$$
$$C(R) = |N| \tag{4.2}$$

Definition 4.6 (Dominance) *Given an indoor space, let R_a and R_b be complete routes covering query keywords, i.e., $q.\psi$. Then, R_a **dominates** R_b if (i) $D(R_a) < D(R_b)$ and $C(R_a) \leq C(R_b)$ or (ii) $C(R_a) < C(R_b)$ and $D(R_a) \leq D(R_b)$.*

Definition 4.7 (Keyword-aware Skyline Routes (KSR) query) *Given an indoor space, a keyword-aware skyline routes query $q = \langle p_s, p_t, \psi \rangle$ where p_s, p_t denote the source indoor point and the target indoor point of the route, and $q.\psi = \langle t_1, \ldots, t_m \rangle$ denotes a set of unique keywords that describes the user preferences. The KSR query returns the set of non-dominated complete routes, i.e., the skyline routes, denoted by S .*

Theorem 4.1 *The problem of solving a KSR query is NP-hard.*

Proof The answer to the classical travelling salesman problem (TSP) is a skyline route because it has the smallest distance. Thus, if KSR query can be solved in polynomial time, the results for the TSP can also be retrieved in polynomial time by returning the skyline route with the smallest distance. Clearly, the problem of solving KSR query is identical to the TSP. Thus, the problem of solving KSR problem is NP-hard.

4.3.2 Limitations of Existing Techniques

The solutions that are proposed in [33] cannot be used to answer KSR queries since they are approximation algorithms. [34, 59, 60] study a variant of TPQ problem called optimal sequenced route (OSR) query. The solutions of these works are not applicable in answering KSR queries as the categories are visited according to a user-specified order. The route planning queries that are investigated in [35, 64, 36] take into account different constraints in determining the optimal route. There solutions cannot be extended to answer KSR queries since they are approximation solutions. [37, 65] study an exact solution for different types of route planning queries. We find that these works have different aims and settings with that of the KSR queries since [37] focuses on finding the optimal route while [65] considers only one keyword per object respectively.

A large body of research can be found [66, 29, 30, 67, 68, 69]. in literature that focus on efficiently finding skyline points in traditional databases. The proposed solution in [31] is not applicable since they consider only one keyword at a time where we focus on multiple keywords in answering a KSR query. [71, 115, 32] focus on retrieving routes with respect to multiple attributes such as distance, driving time and etc. These problems are different from the problem of KSR as they do not take into account any keywords nor distance between them in route construction. Moreover, their solutions are mainly focused on retrieving a set of routes for a given source point and target point that are optimal under any arbitrary liner weighting.

Some studies [73, 74, 72] focus on processing continuous skyline queries on the road networks. These problems find a set of skyline POIs from the same category instead of a complete route covering given keywords. Thus, these

solutions are not applicable to process KSR queries as KSR queries focus on different categories given by the user. Moreover, [76] study a problem that finds a set of skyline routes passes through multiple POIs covering given categories. They propose two approximation algorithms to efficiently solve their problem. Hence, these techniques cannot be extended to answer a KSR query since the KSR query finds the optimal solution.

4.4 GMD Algorithm

Answering a skyline query on the traditional database system is a well-studied problem. Compared to those established solutions, the problem of answering a KSR query has to handle several challenges since routes cannot be assumed to be previously known and also materializing the possible routes is computationally prohibitive due to the huge amount of possible routes with respect to different queries. Hence, we have to perform expensive graph traversals to obtain the all possible complete routes and perform extensive route comparisons to identify the set of non-dominated routes. However, such an exhaustive search is not feasible since the number of possible routes is growing exponentially with the number of query keywords. Thus, we propose an efficient solution called Global Minimum Distance based expansion (GMD) algorithm that traverses the indoor graph in an optimal way such that it prevents generating dominated routes as early as possible. Moreover, we utilize two pruning rules to reduce the search space.

The GMD algorithm constructs candidate routes by progressively retrieving the indoor partitions that cover query keywords. Such an indoor partition is retrieved based on its aggregated distance (denoted by *global minimum distance*) considering both query points, i.e., source and target points. Let P_i

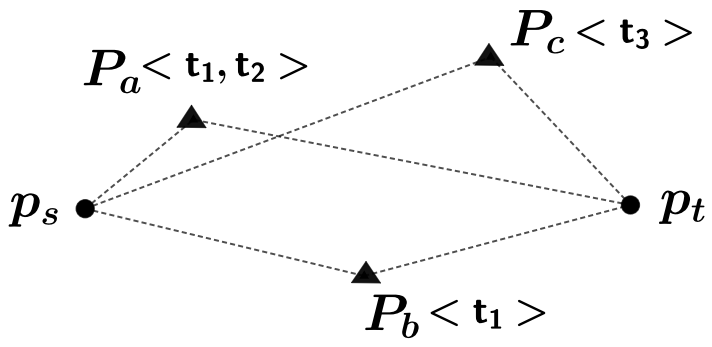


Figure 4.3: Example of constructing a complete route

be an indoor partition and p_s, p_t be the source and target points respectively. Then, the global minimum distance is computed by $dist(p_s, P_i) + dist(P_i, p_t)$ where $dist(p_j, P_i)$ is the indoor distance of the shortest path from indoor point p_j to indoor partition P_i. Whenever such a partition P_i is retrieved, all possible routes through the partition P_i is generated and the current route is extended with reference to the partitions previously retrieved. For example (See Figure 4.3), Let $q = \{p_s, p_t, \{t_1, t_2, t_3\}\}$ be a KSR query and P_a, P_b and P_c be indoor partitions covering query keywords $\{t_1, t_2\}$, $\{t_1\}$ and $\{t_3\}$ respectively. Assume that the order of partitions retrieved is P_a, P_b and P_c with respect to their global minimum distances (Note that, in this study, we consider the indoor distance not the Euclidean distance to compute the global minimum distance). So the partition P_a is obtained first. Since it covers only t_1 and t_2 keywords, a complete route cannot be generated. Next, the partition P_b is obtained and still a complete route cannot be generated. Finally, when the partition P_c is retrieved, we can generate two possible complete routes as follows. (i) $R_x = \{p_s \rightarrow P_a \rightarrow P_c \rightarrow p_t\}$ and (ii) $R_y = \{p_s \rightarrow P_a \rightarrow P_b \rightarrow P_c \rightarrow p_t\}$. Assume that $D(R_x) = 100$ and $D(R_y) = 150$. Then, R_x dominates R_y according to Definition 4.6 since $D(R_a) < D(R_b)$ and $C(R_a) < C(R_b)$. Moreover, Let R_z be a complete route with $D(R_z) = 300$ and $C(R_z) = 1$ (i.e., R_z covers all

query keywords by visiting only one partition). Then, GMD algorithm returns R_a and R_z as skyline routes. Before we present our algorithm, we introduce following pruning rules which utilize the route dominance to reduce a large amount of search space by eliminating dominated routes.

Pruning Rule 5 *For a given KSR query q, let R_a and R_b be routes start from p_s and end at p_i covering the* ***same*** *set of keywords. Then, R_a* ***dominates*** *R_b at point p_i (denoted by $R_a \prec_i R_b$) if $D(R_a) \leq D(R_b)$ and $C(R_a) \leq C(R_b)$.*

Proof In order to prove Pruning Rule 5, we can show that R_a cannot dominate R_b when $D(R_a) > D(R_b)$ or $C(R_a) > C(R_b)$. We use prove by contradiction. Hence, we assume R_a dominates R_b when $D(R_a) > D(R_b)$ or $C(R_a) > C(R_b)$. Let $D(R_a) \leq D(R_b)$ and $C(R_a) > C(R_b)$. Note that, when $D(R_a) == D(R_b)$, R_a cannot dominate R_b. This reveals that our assumption is incorrect. Thus, R_a cannot dominate R_b when $D(R_a) > D(R_b)$ or $C(R_a) > C(R_b)$. We have proven the Pruning Rule 5.

Pruning Rule 6 *Let $R_a = \{p_s, \ldots, p_t\}$ be a complete route and $R_b = \{p_s, \ldots, p_i\}$ be a partially completed route. If $D(R_a) \leq D(R_b)$ and $C(R_a) \leq C(R_b) + 1$. Then, R_a dominates the complete route of R_b. Thus R_b can be pruned.*

Proof To prove Pruning Rule 6, we can show that R_a cannot dominate the complete route of R_b when $D(R_a) > D(R_b)$ or $C(R_a) > C(R_b) + 1$. We use prove by contradiction. Hence, we assume R_a dominates the complete route of R_b when $D(R_a) > D(R_b)$ or $C(R_a) > C(R_b) + 1$. Let $D(R_a) > D(R_b)$ and $C(R_a) \leq C(R_b) + 1$. Note that, when $C(R_a) == C(R_b) + 1$, R_a cannot dominate R_b. This reveals that our assumption is incorrect where R_a cannot dominate the complete route of R_b when $D(R_a) > D(R_b)$ or $C(R_a) > C(R_b) + 1$. Therefore, R_a dominates the complete route of R_b only if $D(R_a) \leq D(R_b)$ and $C(R_a) \leq C(R_b) + 1$. So the partially completed route R_b can be pruned.

Algorithm 7: GMD Algorithm

Data: Query $q = \{d_s, d_t, \psi\}$
Result: Set of skyline routes S

Initialize min-priority queues $\mathcal{Q}, \mathcal{H} \leftarrow \emptyset$;
$kBounds \leftarrow \emptyset$; $S \leftarrow \emptyset$; $candList \leftarrow \emptyset$;
$P_j \leftarrow getMinDistCand\ (\mathcal{H}, candList, true)$;
$R_j^1 \leftarrow generateRoute\ (P_j)$;
$Q.enqueue(R_j^1)$;
while Q *is not empty* **do**
    $R_i^k \leftarrow Q.dequeue\ ()$;
    `// let` $R_i^k = \{d_s, \ldots, d_i\}$
    $P_j \leftarrow getMinDistCand\ (\mathcal{H}, candList, false)$;
    $R_j^1 \leftarrow generateRoute\ (P_j)$;
    `// check for Pruning Rule 5`
    $Q.enqueue\ (R_j^1)$;
    `// check for Pruning Rule 6`
    if $D(R_i^k) < kBounds[k+1]$ **then**
        `// uncovered keywords in the current route`
        $\Psi \leftarrow q.\psi \setminus R_i^k.\psi$;
        if $\Psi == \emptyset$ **then**
            if $d_i == d_t$ **then**
                $flag \leftarrow updateBounds\ (R_i^k, kBounds, S)$;
                `// current route is a non-dominated route`
                if *flag* **then**
                    $S \cup R_i^k$
                end
            end
        end
        else
            $\Delta \leftarrow getAllSubsets\ (\Psi)$;
            foreach *keyword combination* δ *in* Δ **do**
                $PList \leftarrow getAllCandidates\ (\delta, k, candList)$;
                foreach *partition* P_i *in PList* **do**
                    $R_i^{k+1} \leftarrow generateRoute\ (P_i)$;
                    `// check for Pruning Rule 5`
                    $Q.enqueue\ (R_i^{k+1})$;
                end
            end
        end
    end
end
return S

As Algorithm 7 illustrates, first, we initialize two min-priority queues $\mathcal{Q}, \mathcal{H}$ (line 1). The queue $\mathcal{Q}$ is used to maintain the optimal route in terms of route distance and the queue $\mathcal{H}$ maintains the next closest partition with respect to the global minimum distance. And also, we utilize three sets, namely, $kBounds, S$ and *candList* (line 2). The set S and *kBounds* maintain the current skyline routes and their route lengths respectively. The set *candList* is used to store all the partitions that are discovered by Algorithm 8. Initially, we obtain the globally closest partition P_j that covers at least one query keyword by utilizing *getMinDistCand* () function, i.e., Algorithm 8, and generate the optimal route R_j^1 through P_j covering possible query keywords (line 3-5). Since an indoor partition may consist of several doors, we find all possible optimal routes covering given keywords for all door combinations. In order to generate an optimal route inside an indoor partition, i.e., *generateRoute* (), we utilize a *progressive neighbor exploration* [34] based approach that explores keyword nearest neighbor one by one and adding them to route until it covers all keywords. We materialize these routes so that we can efficiently return optimal routes for the same instance, whenever they are needed in later iterations. Then, we insert all routes into the min-priority queue $\mathcal{Q}$. We terminate the algorithm when the Q is empty (line 6). In each iteration, we get the next globally closed partition and generate all possible optimal routes through the partition. Furthermore, we check against the Pruning Rule 5 before we enqueue such a candidate route into the queue $\mathcal{Q}$ (line 8-10). Also, we dequeue the current optimal route R_i^k in the queue Q (line 7) and check whether it can be dominated by another route by utilizing the Pruning Rule 6 (line 11). If it is not dominated, then we get the uncovered query keywords of route R_i^k (line 12). When the candidate route is a complete route, i.e., all query keywords are covered and reached the target point, we update the k bounds. And also,

if the current route is a skyline route then we insert route into the set S (line 13-17). Otherwise, when the route is not a complete route, we generate candidate routes for all possible combinations of uncovered keywords, i.e., Ψ, (line 20-24). For example, if $\Psi = \{t_1, t_2\}$, then $\Delta = \{\{t_1\}, \{t_2\}, \{t_1, t_2\}\}$ (line 19). For each combination, we retrieve the indoor partitions from *candList* that cover the particular keyword combination. Then, we generate routes through each partition and extend the current route (line 23). Then, we check against the Pruning Rule 5 before we enqueue the extended route into the queue $\mathcal{Q}$ (line 24). Finally, we return the set of skyline routes S (line 25).

Next, we explain Algorithm 8 which retrieves the globally closest partition considering the aggregate distance from p_s and p_t. When Algorithm 8 is called for the first time, the root of the inverted VIP-tree is enqueued into the queue $\mathcal{H}$, i.e., when *flag* = *true*. We terminate the while loop either when the queue $\mathcal{H}$ is empty (line 4) or the closest partition with respect to the global minimum distance is found (line 6-8). In each iteration, we dequeue an element and check whether it is (i) a non-leaf-node, (ii) a leaf node or (iii) an indoor partition. If the dequeued element is a non-leaf node, then we enqueue its children (that cover at least one query keyword) with the corresponding global minimum distance as the key value (line 17-20). If the dequeued element is a leaf node, then all the partitions that cover at least one query keyword are selected. Then, each of the selected partitions is enqueued into H for all possible combinations of covered keywords. For example, if the partition P_i covers t_1 and t_2, then three instances of the partition P_i with different covered keywords, i.e., $\{\{t_1\}, \{t_2\}, \{t_1, t_2\}\}$, are inserted into the queue $\mathcal{H}$. This is because a partition which covers two keywords may contribute only one keyword for a complete route. Thus, it is guaranteed to obtain the exact skyline routes. Finally, if the dequeued element is an indoor partition, then

Algorithm 8: *getMinDistCand* (. . .)

```
   Data: min-priority queue H, set of partitions candList, boolean flag
   Result: partition P_k
 1 T ← inverted VIP-tree;P_k ← ∅;
 2 if flag then
 3 |   H.enqueue (T.root, 0);
 4 end
 5 while H is not empty do
 6 |   element ← H.dequeue ();
 7 |   if element is a partition then
 8 |   |   P_k ← element;
 9 |   |   break;
10 |   end
11 |   else if element is a leaf node then
12 |   |   foreach partition P_i of element do
13 |   |   |   if partition.ψ ∩ q.ψ ≠ ∅ then
14 |   |   |   |   Δ ← getAllSubsets (partition.ψ);
15 |   |   |   |   foreach keyword combination δ in Δ do
16 |   |   |   |   |   mDist ← dist (p_n, P_i) + dist (P_i, p_t);
17 |   |   |   |   |   H.enqueue (P_i, mDist);
18 |   |   |   |   end
19 |   |   |   end
20 |   |   end
21 |   end
22 |   else
   |   |   // element is a non-leaf node
23 |   |   foreach childNode C_i of element do
24 |   |   |   if childNode.ψ ∩ q.ψ ≠ ∅ then
25 |   |   |   |   mDist ← dist (d_s, C_i) + dist (C_i, d_t) ;
26 |   |   |   |   H.enqueue (C_i, mDist);
27 |   |   |   end
28 |   |   end
29 |   end
30 end
31 return P_k
```

we break the while loop (line 6-8) and return that partition (line 21).

In this chapter, we utilize VIP-tree [1] which is the state-of-the-art indoor index as our indexing structure. In order to support keyword-based filtering, we modify the VIP-tree by storing inverted files in each tree node. An inverted file consists of a list of all the unique keyword that appears in any indoor partition of that node, and for each keyword, a list of indoor partitions in which it appears. This inverted VIP-tree allows us to efficiently obtain a partition with the minimum distance that consists of at least a given query keyword as it supports both distance and keyword-based filtering.

4.5 Experiments

4.5.1 Experimental Settings

Indoor Space + Keyword Datasets. We use the same indoor venue, i.e., Chadstone Shopping Centre [1] which is the largest shopping centre in Australia, that is used in Chapter 3 as our indoor venue. The D2D graph for the corresponding venue consists of 338 vertices (i.e., doors) and 3847 edges.

In order to obtain the keyword dataset we follow a similar procedure where we crawled data from the websites of major supermarkets (e.g., Coles, Woolworths and *etc.*) as well as major retail stores (e.g., JB Hi-fi, Big W, Chemist and *etc.*) and obtained 140,000 objects along with their keywords. Then, each object was mapped into the particular indoor partition (e.g., retail store) by randomly determining the x and y coordinate of the object inside the particular partition. Furthermore, we obtain a large dataset (denoted as *CHAD2*) by placing a replica of the Chadstone Shopping Centre on top of the original building.

[1] https://www.chadstone.com.au/

Query Generation. In order to generate queries, we took into account a property called objects per keyword (denoted by Ω) which is the number of objects in the indoor space that contains the particular keyword. According to the aforementioned property, we identified 5 different sets of keywords, namely XS, S, M, L and XL. All the keywords in the set XS were obtained by selecting the keywords that have 80 - 120 objects in the indoor space. Similarly, the other keyword sets were obtained by varying the range 180 - 220, 280-320, 380-420 and 480 - 520 objects respectively. Also, for the query point pairs (i.e., source and target points), we obtain three sets of point pairs, namely SH, MI and LO, based on the indoor distance between them (denoted by Δ). The point set SH consists of point pairs that have 180-220 indoor distance between them and, 280-320 and 380-420 for MI and LO respectively. Note that the indoor distances are determined in meters. Finally, a KSR query is generated by randomly selecting the keywords and a pair of points from the corresponding datasets. Accordingly, 50 queries were generated for each experimental setting. Moreover, we followed the same procedure to obtain different query sets for the *CHAD2* dataset.

Competitor. We compare our algorithm GMD with a network expansion based approach, denoted by NE. This approach traverses the indoor graph using network expansion method to find all the possible complete routes. Once it visits an indoor partition it generates all the possible optimal routes through that indoor partition and expands the current route. NE approach utilizes the same progressive nearest neighbor approach which is used by GMD to efficiently generates a candidate route through an indoor partition. Moreover, we employ Pruning Rule 5 in NE to avoid the exhaustive search. Thus, NE ap-

Table 4.1: The parameters used for experiments

Parameter	**Default**	**Range**
Objects per keyword (Ω)	M	XS, S, M, L, XL
# of query keywords ($\|q.\psi\|$)	3	1, 2, 3, 4, 5
Query point distance (Δ)	MI	SH, MI, LO

proach is much better than the naive network expansion based approach.

All algorithms were implemented in C++ and our experiments were conducted on Ubuntu running on an Intel Core i5 @ 3.30GHz and 4GB RAM.

4.5.2 Experimental Results

In all experiments, we use the default settings (see Table 4.1) while varying a single parameter at a time. First, we conduct the set of experiments on the real-world dataset. And for each experiment, we report the average runtime in milliseconds and the average number of intermediate routes generated in query processing.

Varying the number of keywords. Figure 4.4(a) reports the runtimes of algorithms while varying the number of query keywords, i.e., $|q.\psi|$. Note that, under the default settings, each query consist of 3 keywords where each keyword has nearly 300 objects in the indoor space. Even though there are only around 900 related objects in the indoor space, each algorithm deals with 140,000 objects in query processing. We can clearly see that GMD outperforms NE under all configurations. The reason is that GMD can quickly identify a skyline route as it uses the minimum distance approach to retrieve a candidate indoor partition. Also, GMD visits only the candidate partitions that cover the query keywords. GMD answer a KSR query less than 0.3 seconds while NE takes 10 seconds when $|q.\psi| = 3$. Obviously, the performance of both

algorithms decreases when $|q.\psi|$ increases as the number of possible candidate routes increases. GMD performs more than two orders of magnitude faster than NE when $|q.\psi| = 5$. Figure 4.4(b) shows the number of intermediate routes processed by each algorithm while varying $q.\psi$. This clearly explains the reason behind the performance degradation of both algorithms. NE generates more intermediate candidate routes as it uses a network expansion based approach.

Varying the object per keyword. Figure 4.4(c) investigates the runtimes of algorithms varying the number of related objects per keyword, i.e., Ω. Since each query consists of 3 keywords under the default settings, the average number of related objects in the indoor space is 300, 600, 900, 1200 and 1500 respectively. GMD answers a KSR query nearly in one second while NE takes 45 seconds when Ω = XL. Moreover, NE is an order of magnitude worse under all Ω values. As Figure 4.4(d) reports, the number of intermediate routes generated by NE is large compared to GMD. Basically, the number of routes generated by NE is two orders of magnitude higher than GMD. Even though the number of intermediate routes varies in small amounts the runtime drastically increases as the number of computations generating optimal routes inside partitions increases as the number of related objects increases.

Varying the distance. Next, we evaluate the algorithms by varying the distance between query points, i.e., Δ. As Figure 4.4(e) depicts, GMD performs nearly two orders of magnitude better than NE for the short distance query, i.e., Δ =SH, in which GMD takes only 0.1 seconds to answer a KSR query while NE takes 11 seconds. Moreover, GMD outperforms NE by an order of magnitude for medium and long distance queries, i.e., MI and LO respec-

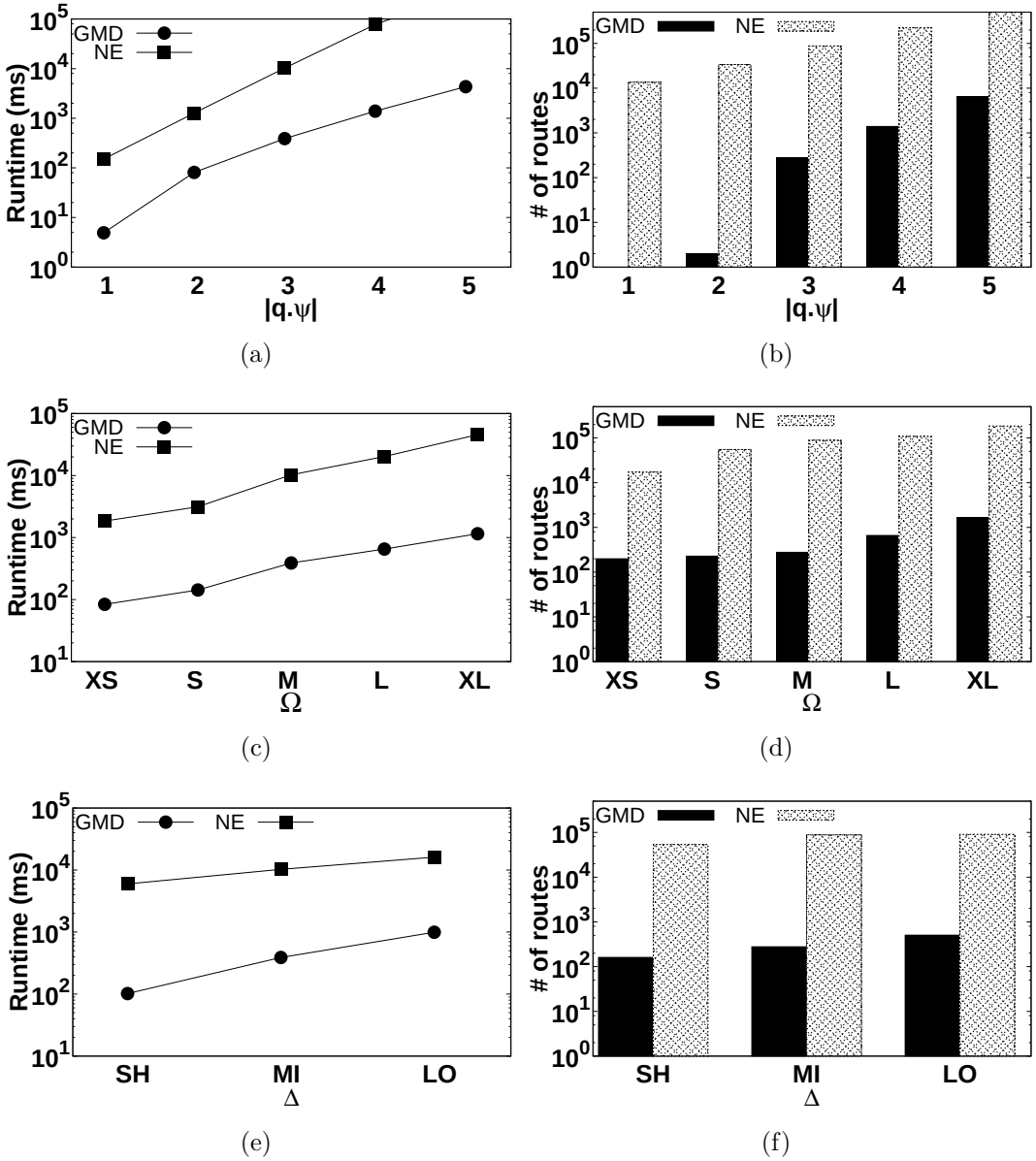


Figure 4.4: Results on the real-world dataset

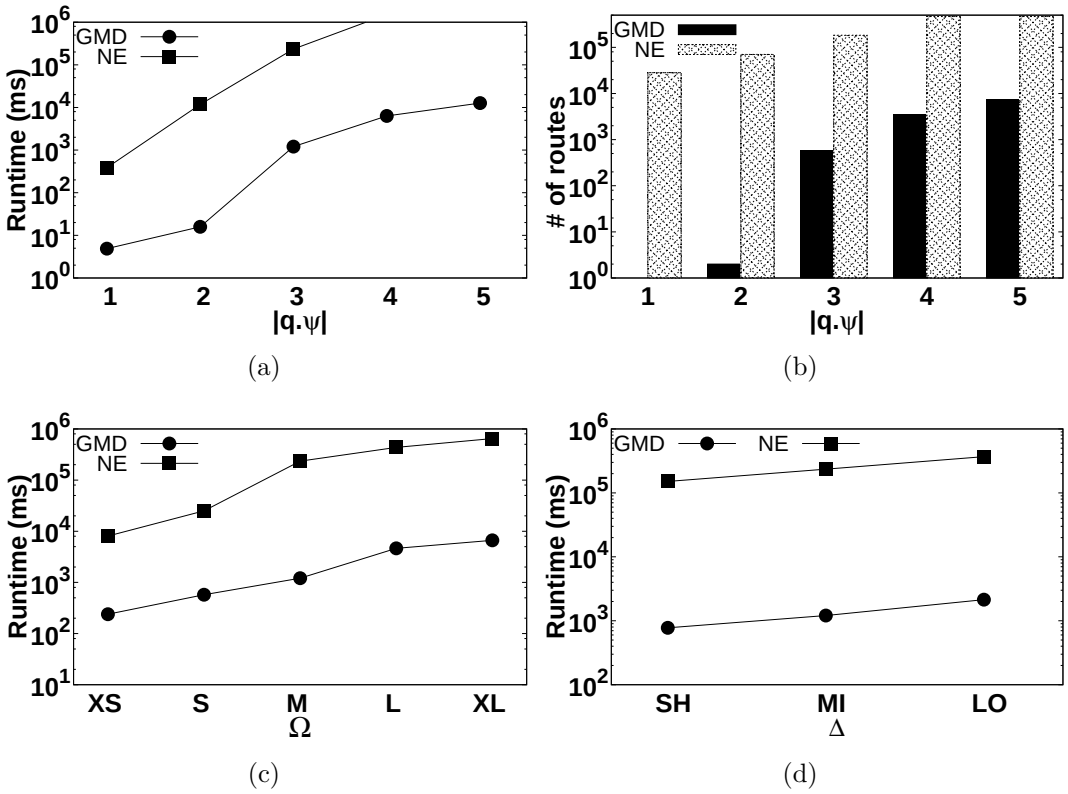


Figure 4.5: Runtime on the *CHAD2* dataset

tively. The performances of algorithms decreases when Δ is increased as the search space increases for both algorithms. Figure 4.4(f) reports the corresponding intermediate routes while varying the distances between the query points. Clearly, the GMD generates a less amount of routes compared to NE as it progressively retrieves the globally closest partition each time.

Experiments on a large dataset. The objective of this set of experiments is to study the scalability of the algorithm. Hence, we use the *CHAD2* dataset which is a replica of the real-world dataset. Note that, this indoor space consists of 240,000 objects and its corresponding D2D graph consist of 676 vertices (i.e., doors) and 7698 edges. Also, we observed that the number of partitions with keywords is much higher in this dataset. Figure 4.5(a) shows the runtimes of algorithms varying $|q.\psi|$. GMD outperforms NE by two orders

of magnitude even when $|q.\psi| = 3$. Also, our algorithm performs reasonably well even for large $|q.\psi|$ values. The performances of both algorithms decreases for this dataset as each algorithm has to visit more partitions. Figure 4.5(b) reports the intermediate routes correspond to the experiment that is performed in Figure 4.5(a). Moreover, as Figure 4.5(c) depicts, the runtimes of algorithms increase when Ω is increased since both algorithms have to generate more routes inside partitions. GMD is two orders of magnitude better than NE when Ω = XL. Finally, Figure 4.5(d) investigates the runtimes of algorithms varying Δ. Clearly, GMD is two orders of magnitude better than NE under all configurations.

4.6 Conclusions

In this chapter, we study a new interesting route planning problem called keyword-aware skyline routes (KSR) query which returns a set of non-dominated routes instead of an optimal route. The dominance of a route over another route is determined based on two attributes, particularly route distance and number of stores/shops visited. KSR queries facilitate the users to find the most suitable route among the skyline routes based on these dimensions. Although the problem of answering a KSR query is NP-hard, it is feasible to design exact solutions for the case where the number of query keywords is limited. The results of the empirical studies on a real-world dataset show the efficiency and the scalability of our exact solution.

Chapter 5

Continuous Detour Queries in Indoor Venues

In this chapter, we investigate continuous detour queries in the indoor space. We propose an efficient client-server framework to continuously answer detour queries in the indoor space. We address the problem by solving two individual subproblems, namely, local and remote computation. We introduce a pre-processing approach for efficient local computation that constructs safe zones for indoor objects. We also propose a best first algorithm for efficient remote computation. Finally, we utilize the outcome of the local and remote computations to answer the continuous detour queries efficiently.

5.1 Overview

Indoor location-based services (LBSs) play an important role in planning daily activities with convenience as people usually spend most of their time inside the indoor venues like office, shopping center, transport facilities, and libraries. Though the efficient processing of variant outdoor LBSs has been

extensively studied over the last decades, with the recent advancement of indoor positioning technologies, the indoor LBSs have received attention from the researchers in the last few years. It has been already shown that the techniques [53, 54, 55, 57, 58] for outdoor LBSs are not adoptable for indoor LBSs due to the unique characteristics of indoor venues. For example, in the road network setting, people move along the roads whereas the buildings are normally organized into partitions with walls, doors, stairs and lifts, and people are free to move inside a partition. Thus, to continue the proliferation of indoor LBSs, it is essential to develop efficient solutions for processing location based queries in indoor venues.

In this chapter, we propose the first solution for continuous detour queries in indoor venues, an essential class of LBSs that allows a moving user to continuously monitor the nearest detour object like an ATM, a printer or a coffee while walking towards a target location in an indoor venue. For example, a user in a library or a shopping centre may want to print or have a coffee before returning to the car park. A detour query returns the indoor object that has the smallest detour distance, where the detour distance is measured as the total indoor distance of an indoor object from the user's current and target locations. However, it is a typical scenario in the indoor venues that a user deviates from the path to the target location via the nearest indoor detour object and the nearest detour object may change. In the library, users may deviate from their current paths to have a look at some books that was not planned or in the shopping centre users may do some window shopping. Sometimes, users may also miss a turn while walking towards the nearest indoor detour object. Thus, it is important to continuously monitor the nearest indoor detour object for every location update of the user walking towards a target location.

Processing a continuous detour query using a detour query for every loca-

tion update of a moving user would incur an extremely high processing overhead. Thus, the efficiency of a continuous detour query processing algorithm for indoor venues highly depends on reducing the number of re-evaluations of the detour queries. To address this issue, in addition to the current nearest indoor detour object, our solution provides a user with additional information so that the user does not need to communicate with the server for the re-evaluation of the detour query for moving within a specific area.

Since the density of the indoor objects is much higher than that of the outdoor space, performing all computations in query time is not feasible for the indoor space. We develop a novel technique to compute the safe zones for indoor objects by exploiting the geometric properties of the hyperbolas and the uniqueness of the indoor space. A safe zone for an indoor object represents an area where it is guaranteed that the indoor object remains the nearest detour with respect to a partition door for a user who is moving towards a fixed target point. Our safe zone computation technique does not depend on any query time parameter and thus, allows us to precompute the safe zones and store them for future access. The precomputed safe zones significantly reduce the computational overhead for identifying the nearest detour objects, and the communication overhead by allowing the user to have the nearest detour objects for a guaranteed area without sending a re-evaluation request of the detour query to the location-based service provider.

The underlying idea of our solution is to divide the problem of finding a nearest detour for a user's current and target locations into two subproblems: identifying the minimum local detour and the minimum remote detour with respect to an indoor partition. For the minimum local detour, using the stored safe zones, we find the indoor object that has the smallest detour distance among all indoor objects in the indoor partition where the user is currently

located. We also develop an efficient algorithm to identify the indoor object that has the smallest detour distance among all indoor objects outside the user's current partition and we consider the identified indoor object as the minimum remote detour. Finally, the indoor object that provides the smallest detour distance between the minimum local detour and the minimum remote detour is selected as the query answer.

The query answer, a grid cell representing an area that may overlap with the safe zones of one or multiple indoor objects along with other additional information are sent to the user. The query answer may change as the user moves but as long as the user resides within the grid cell (denoted as the client-side safe zone), the user can compute the new nearest indoor detour object without communicating with the server. Thus the user's location is again updated to the server for finding the nearest indoor detour object, if the user moves outside the client-side safe zone.

Continuous detour queries have been addressed in the outdoor space [3, 45]. In the road networks, users are restricted to move towards the roads and thus, the existing solution [3] for processing continuous detour queries in the road network is not applicable for indoor scenarios. On the other hand, [45] continuously monitors the nearest detour objects in the outdoor space in presence of obstacles like a building, a river or a fence. Due to the random distribution of the obstacles in the outdoor space, the solution [45] performs all computations during query time and is not suitable for the indoor space.

5.2 Contributions

In summary, the contributions of this chapter are summarized as follows:

- We formulate the problem of answering continuous detour queries in the

indoor space.

- We propose an efficient solution for continuously monitoring the nearest indoor detour object for a user walking towards a fixed target location.
- We develop a novel technique to precompute the safe zones for indoor objects inside a partition.
- We perform an extensive set of experiments and show that our solution performs significantly better than the competitors.

This chapter is organized as follows. Section 5.3 formulates the problem of the continuous detour query and presents a brief discussion about the limitations of existing work, Section 5.4 presents our techniques to answer continuous detour queries in indoor space, Section 5.5 consists of the experiment results and Section 5.6 summarizes this chapter.

5.3 Preliminaries

In this section, we introduce some preliminary definitions and formally define the problem in Section 5.3.1. We present the limitation of existing techniques in both outdoor and indoor spaces in Section 5.3.2. Notations used in this chapter are summarized in Table 5.1.

5.3.1 Problem Definition

Indoor objects

Let $p_i \in \mathcal{P}$ be an indoor point representing an indoor object[1]. The location of an indoor point p_i is given by x and y coordinates.

[1]In this chapter, we use the terms indoor object and indoor point interchangeably.

Definition 5.1 (Detour) *Given a set of indoor points $\mathcal{P}$, the detour $D = \{p_s, p_i, p_t\}$ is a route from a given source indoor point p_s to a given target indoor point p_t going through an indoor point $p_i \in \mathcal{P}$ (denoted as detour point).*

Definition 5.2 (Candidate Partition) *Given a set of indoor points $\mathcal{P}$, an indoor partition I is called a candidate partition only if it consists of at least one indoor point $p \in \mathcal{P}$.*

Definition 5.3 (Local/Remote Detour) *Given a detour $D = \{p_s, p_i, p_t\}$ is called a **local** detour (denoted by D^L) if the detour point, i.e., p_i, belongs to the current indoor partition I where the user resides in. Otherwise, if the detour point lies outside the current indoor partition, it is called a **remote** detour (denoted by D^R).*

Moreover, when the context is ambiguous, we define a local detour by $D^L_{i:j} = \{p_s \rightarrow p_i \rightarrow d_j \rightarrow p_t\}$ where p_i is an indoor point and d_j is a partition door. And a remote detour starts at door d_j by D^R_j.

Definition 5.4 (Detour query) *Given a set of indoor points $\mathcal{P}$, a query $q = \langle p_s, p_t \rangle$ where p_s denotes the current user location and p_t denote the target indoor point. The detour query returns the detour $D = \{p_s, p_i, p_t\}$ subject to:*

$$L(D) = \underset{\forall p_i \in \mathcal{P}}{\textbf{arg min}}\, d(p_s, p_i) + d(p_i, p_t) \tag{5.1}$$

where $d(p_i, p_j)$ is the shortest indoor distance between indoor points p_i and p_j and the $L(D)$ is the indoor distance of the detour.

Definition 5.5 (Continuous Detour query) *Given an indoor space and a moving detour query where p_s continuously changes as a user moves. The continuous detour query continuously find the minimum detour w.r.t the current user point p_s.*

Table 5.1: The summary of notations

Notation	Definition
p_i	an indoor point
q_i	a detour query
I	an indoor partition
$\mathcal{S}_{i:j}$	the splitter between points p_i and p_j
$H_{i:j}$	the region where $D^L_{i:j}$ is the minimum
$D^L_{i:j}$	the minimum local detour from point p_i via door d_j
D^R_i	the minimum remote detour at door d_i
$d(p_i, p_j)$	the indoor distance between points p_i and p_j
$L(D)$	the indoor distance of detour D

5.3.2 Limitations of Existing Techniques

In-Route Nearest Neighbor Queries (IRNN) [38, 2], Path Nearest Neighbor (PNN) [39] and Best point detour [40] study detour problems where the path must be predetermined. Thus, these techniques cannot be extended to answer our problem. In [3], they propose a solution where an order-k shortest path tree is incrementally build. Even though they study a similar problem to ours, we find that their techniques cannot be extended to answer detour queries in indoor space as the indoor detour objects are in Euclidean space. Moreover, [45] investigates detour queries in obstructed space. We find their work have a different aim compared to our problem.

[13, 91, 92, 14] study a variant of nearest neighbor queries called aggregated nearest neighbor which can also be utilized to answer detour snapshot queries. As they do not consider the Euclidean space in the indoor space, we find all these techniques are inapplicable in answering detour queries in indoor space. [65] propose a exact solution to answer route planning queries in indoor space which can be used to answer snapshot detour queries. Their techniques cannot be extended to answer detour queries as they are only efficient when

the number of objects in the indoor space is very small.

5.4 Our Solution

In this section, we develop an efficient solution for processing continuous detour queries in indoor venues. The key idea behind the efficiency of our solution is our novel technique to identify the safe zones, i.e., the areas where indoor objects remain the nearest detours for a user moving towards a fixed target location. We exploit the geometric properties of the hyperbolas, additive weighted Voronoi diagram and indoor partitions to compute the safe zones. A significant advantage of our safe region computation technique is that it does not depend on the query time parameters and thus, allows us to pre-compute the safe zones and index them for use during the query evaluation time. Since the density of indoor objects is much higher than the outdoor space, computing the safe zones during the query evaluation is not feasible for the indoor space. Pre-processed safe zones reduce the computational complexity for finding the nearest detour objects significantly and allows us to provide users with guaranteed areas where the users can locally determine the nearest detour objects without re-evaluating the detour queries for their changed locations. Since a user can freely move inside an indoor partition, the straightforward evaluation of a continuous detour query by using a detour query for every location update of the moving user would incur excessive processing overhead.

Our solution is based on the client-server paradigm for processing the continuous detour queries in the indoor space. The server is the location based service provider and is responsible for evaluating the continuous detour queries, and the client is the user who requests a continuous detour query. The clients send location updates to the server as they move outside the guaranteed area.

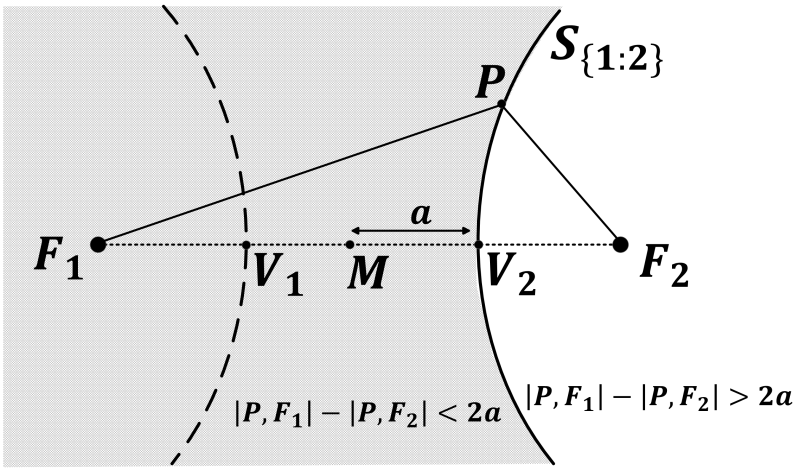


Figure 5.1: Example of a splitter

The server is responsible for maintaining the up-to-date results for the queries with respect to the user's movements. We divide the problem of finding the nearest detour objects into two subproblems, where we compute the minimum local and remote detours separately for each location update. Hence, the query result, i.e., the minimum detour, for a location update can efficiently be determined by utilizing the results of the subproblems. Intuitively, when a movement of a user does not invalidate the previous results of a subproblem, such results can be resued to obtain the minimum detour for the new location of the user. Section 5.4.1 presents the proposed techniques to pre-compute the safe zones, and find the minimum local detour with respect to a user's current and target locations. Section 5.4.2 introduces an efficient algorithm that utilizes the VIP-tree to find the minimum remote detour for a given door of an indoor partition. Section 5.4.3 explains the algorithm for evaluating a detour query and Section 5.4.4 presents the technique for monitoring the nearest detour against the continuous location updates. Finally, Section 5.4.5 discusses a competitive algorithm that performs the *local computation* for a detour query by generating all possible local detours.

5.4.1 Local Computation

In this section, we explain the process of obtaining the minimum local detour for a given user location. We denote this process as the "local computation". We propose an efficient solution that utilize safe zones to quickly identify the minimum local detour for a given indoor point. First, we introduce a pre-processing method to construct such safe zones. Then, we present an indexing method to store the pre-processing results, i.e., safe zones, to utilize in query processing time.

Pre-processing

Naively, the minimum local detour can be determined by generating all possible local detours. Since the indoor space consists of thousands of objects, such an approach is computationally expensive. By exploiting the geometric properties of the hyperbolas, we identify safe zones in the indoor space, where a user's movement does not change the current local detour point that provides the smallest detour distance with respect to a door of an indoor partition. Since these safe zones are independent of the query parameters, we pre-compute and utilize them in the query processing to perform the local computation efficiently. Before we present the pre-processing method, we introduce the following definitions.

As Figure5.1 shows, a hyperbola is a set of points, such that for any point P of the set, the absolute difference of the distances to two fixed points F_1, F_2 (the foci), is constant, i.e., $H = \{P \mid |d(P, F_1) - d(P, F_2)| = 2a\}$. Note that the curve goes through vertex V_2 divides the space into two half spaces where $d(\hat{P}, F_1) - d(\hat{P}, F_2) < 2a$ if the point $\hat{P}$ in the left half space and $d(\hat{P}, F_1) - d(\hat{P}, F_2) > 2a$ if the point $\hat{P}$ in the right half space. Hence, we define such a

curve as follows.

Definition 5.6 (Splitter) *Given a hyperbola of* $|d(P, F_i) - d(P, F_j)| = 2a$. *We identify one of the curves as a splitter that divides the space into two half spaces* $d(P, F_i) - d(P, F_j) < 2a$ *and* $d(P, F_i) - d(P, F_j) > 2a$, *denoted by* $\mathcal{S}_{i:j}$ *where* i *and* j *are the two foci. The corresponding vertex that the curve goes through is called the split point.*

Figure 5.2 depicts an indoor partition I with door d_1. The partition consists of two indoor points p_1, p_2. Assume that the partition I is the only candidate partition in the indoor space. Hence, the minimum detour must be one of the local detours passes through the door d_1 , i.e., $D^L_{1:1} = \{p_s \rightarrow p_1 \rightarrow d_1 \rightarrow p_t\}$ and $D^L_{2:1} = \{p_s \rightarrow p_2 \rightarrow d_1 \rightarrow p_t\}$.

Let $H_{1:1}$ and $H_{2:1}$ be the regions inside the indoor partition where $D^L_{1:1}$ and $D^L_{2:1}$ are minimum detours respectively. When p_s lies in the region $H_{1:1}$, the following inequality must be satisfied.

$$L(D^L_{1:1}) < L(D^L_{2:1})$$

$$d(p_s, p_1) + d(p_1, d_1) + d(d_1, p_t) < d(p_s, p_2) + d(p_2, d_1) + d(d_1, p_t)$$

$$d(p_s, p_1) + d(p_1, d_1) < d(p_s, p_2) + d(p_2, d_1)$$

$$d(p_s, p_1) - d(p_s, p_2) < d(p_2, d_1) - d(p_1, d_1)$$

According to the Definition 5.6, we can construct a splitter (i.e., $\mathcal{S}_{1:2}$) by selecting the indoor points p_1, p_2 as foci and $2a = d(p_2, d_1) - d(p_1, d_1)$. The most important step in constructing a splitter is to determine the split point correctly. Let point x be the split point and point c be the center. Then the

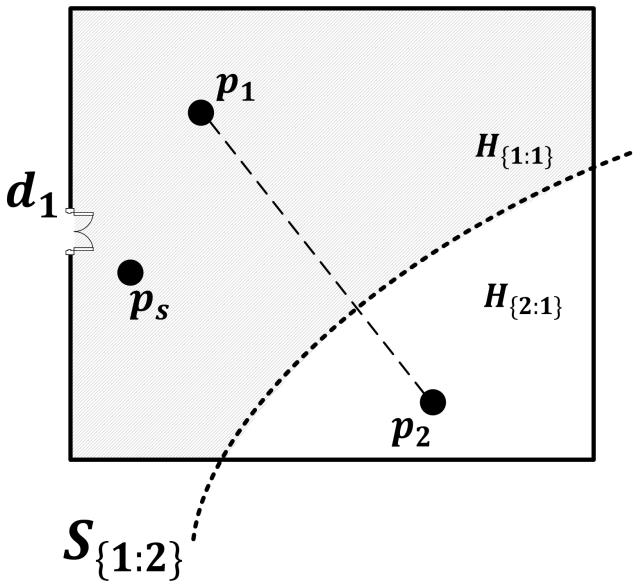


Figure 5.2: Example of a region divided by a splitter

split point is determined as follows,

$$\begin{aligned} & d(p_1, x) - d(p_1, c) = a \\ & d(p_1, x) - \frac{d(p_1, p_2)}{2} = \frac{1}{2}\Big(d(p_2, d_1) - d(p_1, d_1)\Big) \\ & d(p_1, x) = \frac{1}{2}\Big(d(p_1, p_2) + d(p_2, d_1) - d(p_1, d_1)\Big) \end{aligned} \tag{5.2}$$

Once the splitter $\mathcal{S}_{1:2}$ is constructed, space is divided into two half spaces $d(p, p_1) - d(p, p_2) < 2a$ (i.e., the shaded area in Figure 5.2) and $d(p, p_1) - d(p, p_2) > 2a$ which is basically the required $H_{1:1}$ and $H_{2:1}$ regions respectively. Intuitively, these two regions act as safe zones where we can guarantee that the minimum local detour does not change until the current location of the user is within a particular region. Moreover, as the Equation 5.4.1 depicts, the splitter $\mathcal{S}_{1:2}$ is independent of the query parameters. Hence, these two regions can be pre-computed and utilized in query time to quickly identify the minimum local detour that goes through the door d_1 with respect to a location update inside the partition I.

Furthermore, for the candidate partitions with indoor points more than two, we need to construct splitters for each possible pair of indoor points to determine such safe zones. Note that these regions are AW Voronoi cells with additive weight $d(d_i, p_i)$. Hence, in the pre-processing approach, we generate AW Voronoi diagram for each partition door d_i. In query processing, we can easily determine the minimum local detour that goes through a particular door by looking at the corresponding AW Voronoi cell with respect to the current user location. For the indoor partitions that have more than one door, AW Voronoi diagram for each door d_i is created with the additive weight $d(d_i, p_i)$. For such a partition, the local computation is done as follows. First, the minimum local detour points with respect to each door is obtained using corresponding AW Voronoi diagrams. Then, these results are evaluated to determine the minimum local detour point. Since the number of doors of a candidate partition is small in real-world applications, i.e., at most 3-4 doors, the local computation can be done efficiently.

Safe zone Indexing

As we have already stated, AW Voronoi diagrams for each candidate partition is determined by constructing the splitters between the indoor points. Since these Voronoi cells are formed by curved splitters, such an AW Voronoi diagram cannot be easily indexed unless they are approximated using polygons. But it incurs inaccurate results in query time. Hence, we accompanied an approach that index AW Voronoi diagrams by utilizing the grid data structure. In our grid index, the space of the corresponding indoor partition is subdivided into $2^n \times 2^n$ grid cells (where $n > 0$) as shown in Figure 5.3. Then in each grid cell, we store the splitters that overlap with the cell. To record these overlapping splitters in each grid cell, we used conceptual tree-based grid access method [4]

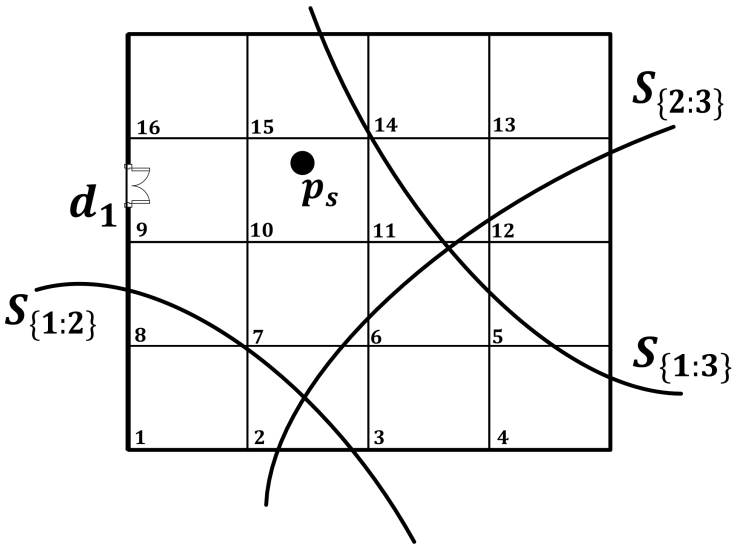


Figure 5.3: Example of a 4×4 grid

where we consider the root of the conceptual tree is a rectangle covering the whole indoor partition. Then the root cell is divided into four equal grid cells that represent the next level of the tree. The process continues until each entry of the leaf level represents one grid cell. During the process, for an intermediate cell, we take into account the splitters overlaps with the cell and only these splitters are considered when marking the child cells of the particular cell. By doing this, we significantly improve the process of constructing the AW Voronoi diagrams for a given partition.

For example, Figure 5.3 shows the AW Voronoi diagram (which is a 4×4 grid) for door d_1 of the partition I . The grid cell 2 will be marked with $S_{1:2}$ and $S_{2:3}$ as they overlap with the particular grid cell. Similarly, grid cell 11 is marked with $S_{1:3}$ and $S_{2:3}$. The grid cells $1, 3, 9, 13, 16$ remains empty as they are not overlap by a splitter. Which indicates that these grid cells are completely inside AW Voronoi cells. Thus, the minimum local detour point (i.e., the corresponding indoor point that the Voronoi Cell belongs to) for a user location within these grid cells can be quickly determined. For the grid cells with overlapping splitters, we have to check with the overlapping splitters

to determine the minimum local detour point accordingly.

5.4.2 Remote Computation

In this section, we present an efficient approach to compute results for our second subproblem which is determining the minimum remote detour with respect to the user's location (denoted as "remote computation"). Intuitively, for any user location inside an indoor partition, the minimum remote detour must be a remote detour that starts from one of the partition doors. Hence, we propose an algorithm *getRemoteDetour()*, i.e, Algorithm 9, based on best first search that retrieves the minimum remote detour from a given door to the query target point p_t. This algorithm accesses the VIP-tree components, i.e, tree nodes, partitions and indoor points, based on the smallest aggregated indoor distance from the given door and the query target point to retrieve the minimum remote detour point. After obtaining the minimum remote detour points for each partition door, the minimum remote detour with respect to the user location can be readily determined. Moreover, if the next location update is within the same partition, then we can reuse these remote detour results to determine the minimum remote detour for the new user location. As Algorithm 9 illustrates, we traverse each level of the VIP-tree starting from the root node (line 2) and compute the detour costs for tree nodes at each level. We terminate the algorithm when the min-priority queue $\mathcal{Q}$ is empty (line 3) or the minimum detour point is found (line 5-6). We enqueue each element with the detour distance as its key value. In each iteration we dequeue the element with the minimum key value, i.e., smallest detour distance. If the dequeued element is a node we enqueue all the child nodes (line 16-21). If the element is an indoor partition then we enqueue all the indoor points inside the partition. Note that, an indoor point may be enqueued multiple times into

Algorithm 9: *getRemoteDetour*(d_k, p_t)

Data: VIP-tree $\mathcal{V}$, door d_k, indoor point p_t
Result: minimum remote detour D^R

$p \leftarrow \emptyset$; $D \leftarrow \emptyset$
$\mathcal{Q}.enqueue\ (\mathcal{V}.root, 0)$;
while $\mathcal{Q}$ *is not empty* **do**
    $element \leftarrow \mathcal{Q}.dequeue()$;
    if *element is a point* **then**
        $p \leftarrow element$;
    end
    else if *element is a partition* **then**
        foreach *point* p_a *in element* **do**
            foreach $\{d_i, d_j\}$ *door combination* **do**
                $minCost \leftarrow d(d_k, d_i) + d(d_j, p_t) + d(d_i, p_k) + d(p_k, d_j)$;
                $\mathcal{Q}.enqueue\ (p_a, minCost)$;
            end
        end
    end
    else
        `// element is a tree node`
        foreach *child-node* N *of element* **do**
            $minCost \leftarrow d(d_k, N) + d(N, p_t)$;
            $\mathcal{Q}.enqueue\ (N, minCost)$;
        end
    end
end
$D^R \leftarrow \{d_i, p, p_t\}$
return D^R;

the queue as the detours through different door combination are possible (line 8-15). When the dequeued element is an indoor point, we terminate. Finally, we return that indoor point and its detour distance (line 24).

5.4.3 Query Processing

Now we proceed to explain our solution that utilize the results of the two subproblems, i.e., local and remote computations, to determine the minimum detour for a given query. When the sytem is initiated, the AW Voronoi diagrams of all the partitions are loaded into the server memory. Thus, the AW Voronoi diagram for an indoor partition can be accessed efficiently in the query processing. As Algorithm 10 illustrates, first, we initialize I with the current partition (line 3). Next we compute the remote detours for each door of the partition I (line 5). Meanwhile, we update the minimum remote detour with respect to the current user location by selecting the remote detour with the smallest detour distance (line 6-7). Then, we check whether the current partition, i.e., I, is a candidate partition (line 10). If so we must perform the local computation to obtain the minimum local detour point. Thus, we utilize each of the AWVDs of partition doors to compute the minimum local detour. Initially, the grid cell that the current user location p_s lies is identified. Then, the splitters overlaps with the corresponding grid cell are retrieved (line 14). Also, we store these splitters using the list $sList$, (in Section 5.4.3, we explain the purpose of storing these splitters). Next, the minimum local detour point is obtained by evaluating the overlapping splitters (line 12-14). After determining the local detour points per door, we generate the corresponding local detours to find the one with minimum detour distance among them (line 15-17). Note that, the shortest path distances from door d_i to target point p_t, i.e., $dt(d_i, p_t)$, must be determined to compute the local detour distances.

Then the safe zone for the user is constructed (line 21). Note that the safe zone for the user is computed using the indexed safe zones of the indoor points and the detail of the construction of such a safe zone for the user is discussed in Section 5.4.3. The minimum detour is determined by selecting the minimum of the remote and local detours (line 22). Finally, the minimum detour and the safe zone are returned (line 23).

Safe zone Construction for Users

In order to reduce the communication cost, we send a safe zone such that the client device can monitor the query results by itself without contacting the server. Even though the actual safe region for a minimum local detour via a door is the corresponding AW Voronoi cell, we assign the area of the grid cell that user located in. The reason behind this is to reduce the workload at the client-side by sending only a small number of boundaries to monitor (Note that an AW Voronoi cell may consist of a large number of boundaries.) Since a grid cell may belong to different AW Voronoi cells, the overlapping splitters of the particular cell are sent to the user to determine the minimum local detour point for the next location update without communicating the server. Moreover, the distances of the shortest paths from each partition door are also sent to assist the client-side local computation.

In addition to that, the remote detours for each door are sent to the user for the client-side remote computation. Thus, the client device can compute the query result for the next location update without communicating with the server if the user is still inside the safe zone. Note that, for a non-candidate partition, only the remote computation is required. Thus, we assign the whole space of the indoor partition as the safe zone and send the materialized remote detours to support the client-side remote computation. Thus, the client

device can determine the minimum detour for any location update within the particular partition without contacting the server.

Algorithm 10: Query Processing

```
   Data: VIP-tree V, a query q = {p_s, p_t}
   Result: Minimum Detour D, Safe zone safe
 1 D^R ← ∅; D^L ← ∅; sList ← ∅;
 2 I ← current partition;
   // Remote Computation
 3 foreach d_i door of I do
 4 |   D_i^R ← getRemoteDetour(d_i, p_t) ;                    // Algorithm 9
 5 |   if d(p_s, d_i)+L(D_i^R) < L(D^R) then
 6 |   |   D^R ← D_i^R ;                  // minimum remote detour w.r.t p_s
 7 |   end
 8 end
   // Local Computation
 9 if I is a candidate partition then
10 |   foreach d_i door of I do
11 |   |   Cell ← the corresponding grid cell;     // using AWVD of door d_i
12 |   |   sList ← sList ∪ the set of splitters overlaps with Cell;
13 |   |   p_j ← local detour point for d_i
14 |   |   L(D_i^L) ← {p_s → p_j → d_i → d_t};
15 |   |   if L(D_i^L) < L(D^L) then
16 |   |   |   D^L ← D_i^L ;                 // minimum local detour w.r.t p_s
17 |   |   end
18 |   end
19 end
20 construct safe ;                                          // Section 5.4.3
21 D ← min{D^R, D^L};
22 return D, safe;
```

5.4.4 Continuous Monitoring

The result of the query needs to be continuously monitored since the user is continuously moving and user movements may change the query result. Naively, the server can recompute the query result at each time the server receives a location update from the user. Since the indoor venues consist of thousands of indoor points, these re-computations are very expensive. In the experiments, we show the performance difference between this naive approach

and our solution. As we already stated, some initial computations for an indoor partition can be reused until the user leaves the particular partition. Hence, we materialize the minimum remote detours (line 5 in Algorithm 10) and shortest path distances (line 16 in Algorithm 10) for each door of the current indoor partition. Then, query result computation for any location update about to happen inside the current partition can be handled efficiently.

In continuous monitoring, first, the server checks whether the user still in the same partition so that the server can reuse the materialized data to determine the minimum detour. The server uses the materialized remote detours to compute the minimum remote detour for the location update. Then the server accesses AW Voronoi diagrams to figure out the minimum local detour points for each door of the partition and determines the minimum local detour. Finally, the server sends the safe zone along with the minimum detour to reduce the communication cost occur due to frequent location updates. With the help of the safe zone constructed for the user, the client device can monitor the query result without communicating the server.

In the client side, we assume that the client device is capable of storing all the information that is sent by the server, i.e., the remote detours, the shortest path distances and the safe zone, and computing minimum detour for any user movement within the safe zone. Once the user moves outside the safe zone, it sends a location update to the server. Then the server computes the query result for the new location and send the result along with the safe zone.

For example, Figure 5.4 shows a 4×4 grid of a candidate partition with door d_1. The area shaded in grey is the AW Voronoi cell of the indoor point p_1 and white area is the AW Voronoi cell of the indoor point p_2. Let the user is at p_{s_1} position. Thus, the area of the grid cell 1 is allocated as the safe zone for the user. Assume that the user moves to p_{s_2}. Since the user is still within the

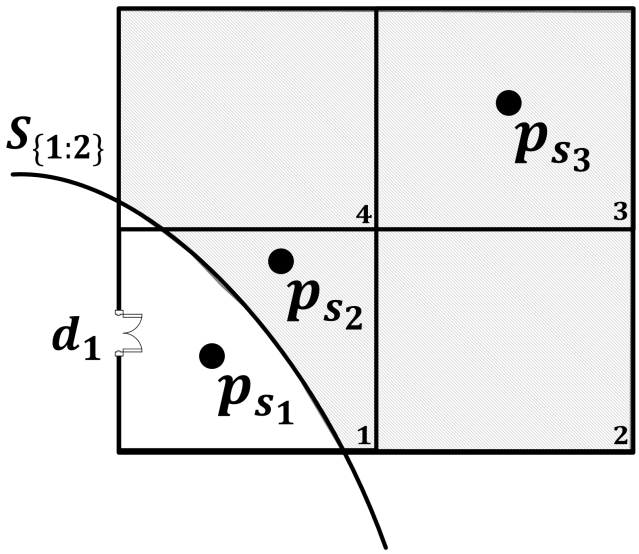


Figure 5.4: Example of continuous monitoring

safe zone, the client device will determine the local minimum detour point as p_1 by evaluating the splitter $S_{1:2}$. Moreover, the client device will perform the remote computation by utilizing materialized data to determine the minimum detour without communicating with the server. For the next movement, i.e., p_{s_3}, the client device will send a location update to the server since the user has leaf the safe zone.

5.4.5 Local Computation without AWVDs

As indoor venues consist of a considerable number of indoor points, most of the indoor partitions are highly populated with indoor points. Hence, a large amount of possible local detours must be generated to determine the minimum local detour. We develop a competitive approach called **local** which is an improvement of the naive approach. Similar to our approach, the local approach utilizes the materialized data such as the remote detours and the distances of the shortest paths from each partition door such that they are utilized in the next location updates happen within the current partition. Besides, the

local approach generates all possible local detours to determine the minimum local detour and send only the result to the user. Thus, the communication overhead remains the same as the naive approach. In the empirical study, we show that our solution performs much better than this approach.

5.5 Experiments

5.5.1 Experimental Settings

Indoor Space Datasets.

We used two indoor space datasets for our experiments. One of them is the real-world dataset [111] of the largest shopping centre in Australia. The dataset consists of over 300 indoor partitions that are spread over 4 levels. This dataset was manually converted into machine-readable indoor venues and the sizes of indoor partitions (e.g., rooms, hallways) were determined using OpenStreetMap [2]. We denote this dataset by CHAD. The D2D graph for this indoor space consists of 338 vertices (i.e., doors) and 3847 edges. We synthetically generated five indoor object datasets that have 1K, 2K, 3K, 4K and 5K indoor points, respectively. Each indoor point was randomly selected inside a partition of the indoor space.

The other dataset is a replica of the CHAD dataset (denoted as CHAD-2). It was obtained by placing a replica of Chadstone Shopping Centre on top of the original building. This dataset consists of 678 rooms and the D2D graph for this indoor space consists of 676 vertices (i.e., doors) and 7698 edges. We again generated five indoor object datasets for this indoor space by randomly selecting 5K, 10K, 15K, 20K and 25K indoors points, respectively. Moreover,

[2]https://www.openstreetmap.org/

we generate 100 queries for each experimental setting. The source and target indoor points of each query are randomly determined.

Trajectory Generation.

We created synthetic trajectory datasets for all our experiments. To generate a trajectory, we start from the query source point and randomly pick an indoor point inside a candidate partition. By doing this, we make sure that the user trajectories pass through candidate partitions. Also, these random points are selected in a way that the trajectory leads towards the query target point to demonstrate the real-world scenarios. After determining a random indoor point, we let the user moves towards that point with a particular speed. We continued this for 500 timestamps, by determining a new random point similarly as the user reaches one. The walking speeds of the users are determined as follows. We chose 0.5, 1.5 and 2.5 meters per timestamp as the user speeds in generating the different trajectory datasets. Furthermore, we denote these speeds by slow, medium and fast respectively in the later discussions.

Competitors.

We compare our algorithm with two competitors. First one is the naive approach which computes the query result from scratch for each location update. We denote this by "naive". Our second competitor is the approach mentioned in Section 5.4.5 denoted by "local" which is an improvement of the naive approach that does only the local computation naively for each location update.

All algorithms were implemented in C++ and our experiments were conducted on Ubuntu running on an Intel Core i5 @ 3.30GHz and 4GB RAM.

Table 5.2: The parameters used for experiments

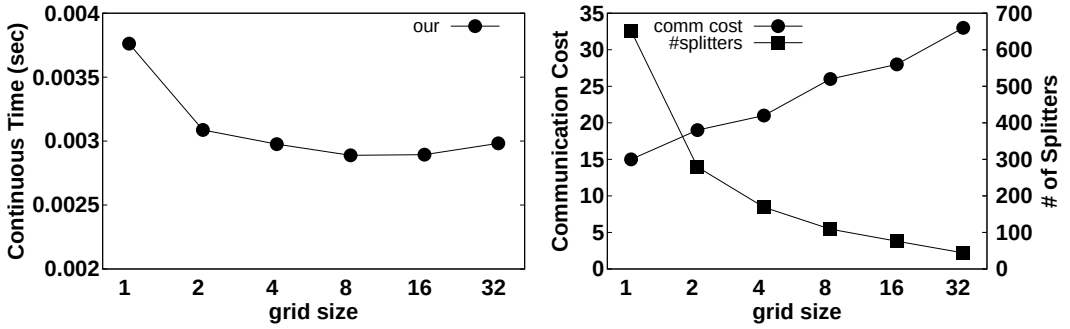
Parameter	**Default**	**Range**
# indoor points	3K	1K, 2K, 3K, 4K, 5K
speed	medium	slow, medium, fast
grid size	8	1, 2, 4, 8, 16, 32

5.5.2 Experimental Results

In all the experiments, we use the default settings (see Table 5.2) while varying a single parameter at a time. First, we conduct the set of experiments using CHAD. Then for CHAD-2 which is the replica of the real-world dataset. Moreover, for each experiment, we report the average continuous time in milliseconds.

Varying the grid size

As mentioned in Section 5.4.1, we use the grid indexing structure to index the additively weighted voronoi diagrams (AWVDs). And these voronoi cells are utilized in query time for efficient local computation. As Figure 5.5(a) illustrates, our solution perform well when grid size is 8 which is less than 0.003 seconds. Figure 5.5(b) report the communication cost and the number of splitters which are basically the average number of times that the client device communicate with the server and the average number of splitters monitored by the system respectively. As Figure 5.5(b) illustrates, the communication cost can be reduced by selecting the grid size as 1. The reason is that the whole space of indoor partition is assigned as the safe zone. Even though small grid sizes give low communication cost as large area is assigned as safe zones, the client side computational cost increases since the number of splitters monitored by the client device increases accordingly. For example, even though the average communication cost for grid size 1 is 15, the client device has to



(a) (b)

Figure 5.5: Varying grid size on the CHAD dataset

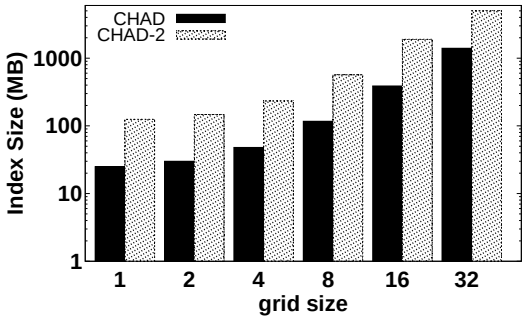
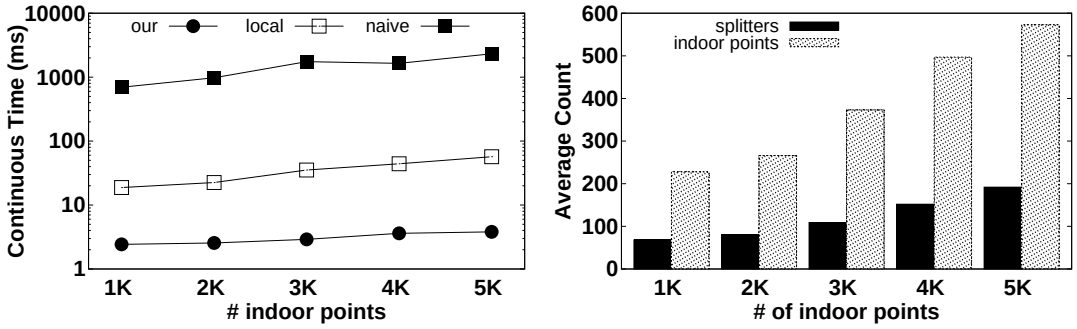


Figure 5.6: Indexing Cost

monitor more than 650 splitters in the local computation. We selected the grid size 8 as the default gird size as it gives the best performance for our solution. For grid size 8, the communication overhead is 25 which is 20 times better than the competitors. And also, the system has to monitor nearly 100 splitters per detour query.

Moreover, we report the indexing cost of AW Voronoi diagrams in megabytes. As Figure 5.6 illustrates, the amount of memory required to store the AW Voronoi diagrams increases as the grid size increases because the number of cells increases exponentially. But the required memory spaces for small grid sizes are feasible. For the default grid size which is 8, the AW Voronoi diagrams of CHAD and CHAD-2 datasets need only 117MB and 567MB memory space respectively.



(a) (b)

Figure 5.7: Varying the number of indoor points

Varying the number of indoor points

This set of experiments is done to evaluate all the algorithms on the CHAD dataset. First, we investigate the continuous time of the algorithms by varying the number of points (i.e., indoor objects) in the indoor venue. As Figure 5.7(a) shows, the continuous times of all algorithms increase as the number of indoor points are increased. The reason is both remote and local detour computation costs increase as the number possible detours increases. Even though our solution does not consider all possible detours in local computation, still the continuous time increases accordingly as the number of splitters to be monitored increases. For the default settings, our algorithm is 15 times better than the local approach while three orders of magnitude better than the naive approach. As the competitors do not use the safe zone concept, in terms of the communication overhead we are always better since they have to communicate the server to get the results. The continuous time of naive drastically increases as it does both remote and local computation for each location update. Clearly, our solution outperforms the competitors under all the settings. Figure 5.7(b) reports the average number of indoor points accessed by the local approach and the average number of splitters monitored by our approach. It is obvious that both values increases as the indoor points in the indoor venue increases.

Moreover, the reported results clearly explain the performance degradation of both our and local approaches in the experiments shown in Figure 5.7(a).

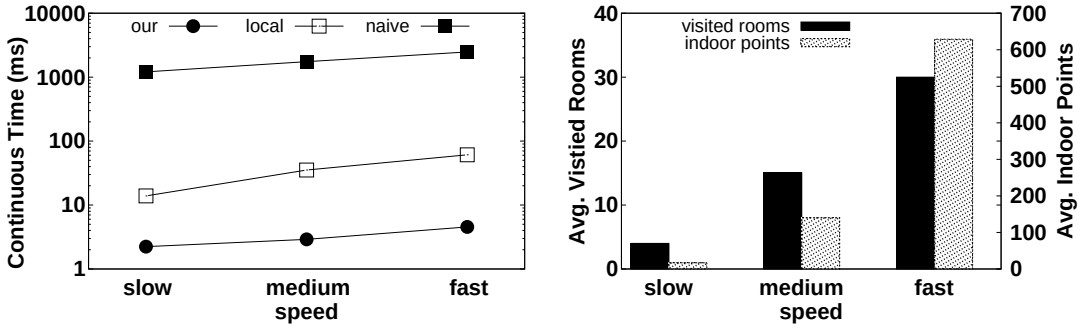
Varying the speed

Next, we evaluate the algorithms by varying the walking speed of the user. As Figure 5.8(a) shows, the continuous time increases for all algorithms when the speed is increased. It's obvious for our and local solution that the continuous time increases as the user quickly leaves partition due to the higher walking speed. Note that, for our solution, the user stays inside a safe zone only for a small amount of time. The reasons for this performance degradation of the naive solution is that the user is visiting more candidate partition when the speed is increased. Thus, local and remote detour computations increases.

Figure 5.8(b) reports the average number of rooms visited by the user and the average number of indoor points accessed by the local algorithm by varying the speed of the user. When the user's speed is fast, the user visits 30 rooms while only 5 rooms when speed is slow. As we mentioned earlier, clearly the continuous times of all algorithms increase as the speed increased since the user is quickly leaving partitions. Also, the number of candidate partitions visited by the user has increased as she visits more rooms. Thus, the continuous times of both our and local approaches increases as the number of indoor points accessed and the number of splitters monitored increase.

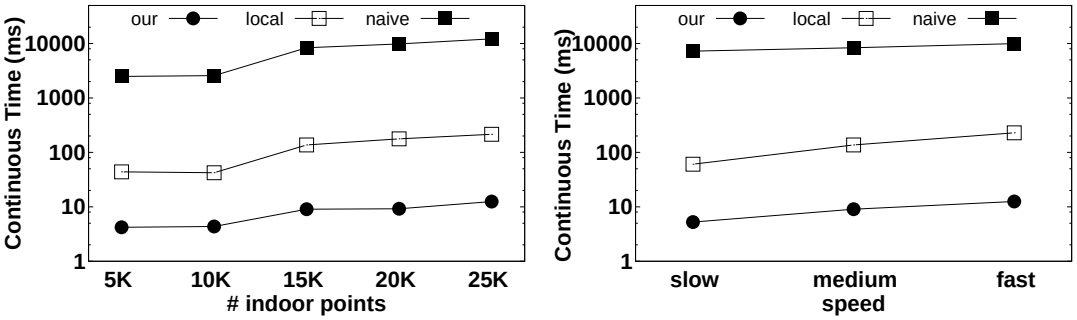
Experiments on a large dataset.

The objective of this set of experiments is to study the scalability of the algorithm. Hence, we use the CHAD-2 dataset. Since this dataset has a large number of indoor points, the number of candidate partitions for this dataset is higher than the real-world dataset. Figure 5.9(a) shows the continuous times



(a) (b)

Figure 5.8: Varying the speed of the user



(a) (b)

Figure 5.9: Results on the CHAD-2 dataset

of algorithms varying the number of indoor points. The performances of all the algorithms decrease when the number of indoor points is increased. Clearly, our approach is three order of magnitude better than naive solution when the dataset consists of 25K indoor points. And also 20 times better than the local approach. Moreover, Figure 5.9(b) reports continuous times by varying the user speed on CHAD-2 dataset. Under all the settings, our approach performs better than competitive approaches. The performance of all algorithms has decreased compared to the results on CHAD dataset due to a large number of indoor points and candidate partitions. Note that, our approach is still able to answer a query in a reasonable time. These results conclude that our approach has good scalability.

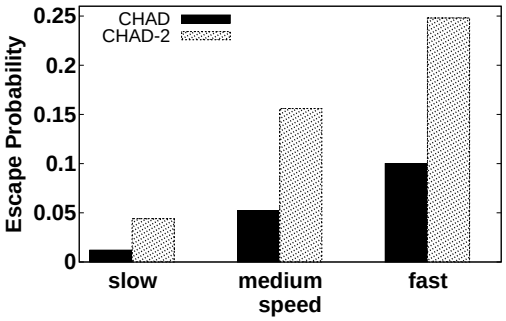


Figure 5.10: Escape Probability

Effectiveness of safe zones

Figure 5.10 reports the escape probability of our solution varying the users' speed. The escape probability is determined by dividing the number of times user communicate with the server by the total number of movements. As expected, the escape probability increases with the user speed. The reason is, the users leave safe zones very quickly due to the speed. The escape probability is 0.1 in CHAD dataset while 0.25 in CHAD-2 dataset when the user is moving fast. The reason behind the high escape probability of the CHAD-2 dataset is that the number of candidate partitions in the CHAD-2 dataset is large. Hence, the users pass through many candidate partitions in their trajectories. The results conclude that the escape probability is small and the proposed solution is effective in real-world applications.

5.6 Conclusions

In this chapter, we propose an efficient solution for continuously answering detour queries in the indoor space. We address the problem by solving two individual subproblems, namely, local and remote computation. First, a pre-processing approach is introduced for efficient local computation that constructs safe zones for indoor objects. Then a best first search algorithm is

presented to efficiently compute a remote detour for a given door of an indoor partition. Finally, we introduce a client-server framework that utilize the outcome of the local and remote computations to answer the continuous detour queries efficiently.

The results of the empirical studies show that our approach outperforms the naive approach by at least three orders of magnitude while average 15 times faster than the improved naive approach. Also, our approach incurs less communication overhead due to the safe zone approach in which it is at least 20 times better than the competitors.

Chapter 6

Continuous Monitoring of Range Spatial Keyword Query over Moving Objects

In this chapter, we propose an efficient solution for processing continuous range spatial keyword queries over moving spatio-textual objects (namely *CRSK-mo* queries). The key idea of our approach is to exploit the spatial and textual upper bounds between queries and objects to form *safe zones* (at the client-side) and *buffer regions* (at the server-side), and then use these bounds to quickly prune objects and queries through smart in-memory data structures. Our research reported in this chapter appears in [116].

6.1 Overview

The proliferation of GPS-enabled mobile devices, and the huge popularity of location based social networking sites (LBSN, e.g., Foursquare, Yelp) have facilitated the generation of a large volume of geo-textual (or spatio-textual)

datasets which form the basis of many emerging location based services (LBS). One of the important and popular forms of queries in LBS is the spatial keyword query: given a set O of spatio-textual objects where each $o \in O$ is described by its location and a set of keywords, a spatial keyword query q finds objects that meet the requirements of the query in terms of both spatial proximity and textual similarity. The spatial keyword queries have been extensively studied in different contexts that include range query, k nearest neighbour query, top-k query, and publish-subscribe query. A comprehensive study of these queries can be found in [43]. However, to the best of our knowledge, we are the first to study continuous monitoring of *moving* spatio-textual objects for thousands of continuous (long running) queries in real time. Next, we present our motivation for studying this problem.

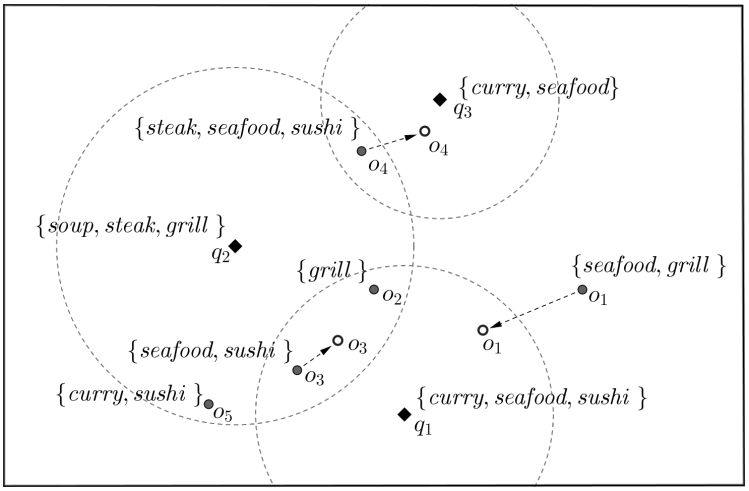
6.1.1 Motivation

Consider the example of a fast food chain that wants to continuously monitor the potential customers to send them targeted advertisements and deals. Potential customers of a fast food outlet are the users that are close to it and whose preferences (keywords) match the menu of the restaurant. The fast food chain may want to continuously monitor all such potential customers to increase their sale. Similarly, a supermarket may want monitor the people who are close to it and are looking for products sold at the supermarket. These people may be attracted by sending e-coupons or personalized deals. Spatial keyword queries are also important in other domains. For example, in an emergency scenario, hospitals could monitor the locations of health workers or volunteers, and send requests to those who are nearby and whose expertise match with the required expertise. In all of the above scenarios, we need to continuously monitor moving spatio-textual objects (e.g., customers or users)

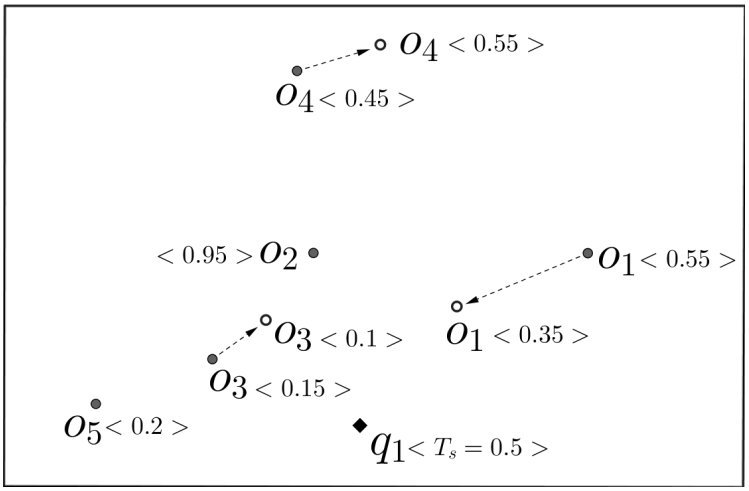
for multiple long running range queries with respect to query objects (e.g., facilities such as restaurants or hospitals). We call these queries continuous range spatial keyword queries over moving spatio-textual objects (or *CRSK-mo* queries). Next, we provide an example of such queries.

Example Figure 6.1(a) shows three range queries q_1, q_2, and q_3 and five spatio-textual moving objects (users) o_1, o_2, o_3, o_4, and o_5. Assume that a query wants to track every user whose current location is inside the query range, and who has at least one common keyword with the query keywords. In this case, at time t_1, the *CRSK-mo* query returns $RS_{q_1}^{t_1} = \{o_3\}$, $RS_{q_2}^{t_1} = \{o_2, o_4\}$ and $RS_{q_3}^{t_1} = \{o_4\}$ as the result sets of q_1, q_2, and q_3, respectively. Now, at time t_2, objects o_1, o_3, o_4 move to new locations as shown using small blank circles. Thus, the query results are updated as $RS_{q_1}^{t_2} = \{o_1, o_3\}$, $RS_{q_2}^{t_2} = \{o_2\}$ and $RS_{q_3}^{t_2} = \{o_4\}$.

In the above example, we explain the concept of *CRSK-mo* queries by using *boolean range queries* [95, 98, 27] where an object is a result of a query if it is within a specific range and matches a keyword criteria (e.g., at least one keyword matches). A general and often preferred approach is to define the relevance of an object to a query using a relevance score that combines both spatial proximity and textual similarity between the query and object. In this case, a query user may set a *threshold score* T_s and every object with relevance score less than or equal to the given threshold score is returned as an answer. Figure 6.1(b) shows an example where the objects' relevance scores are shown next to each object and the movement to a new location is shown by an arrow. Thus, the results of the query q_1 for the two timestamps are $RS_{q_1}^{t_1} = \{o_3, o_4, o_5\}$, and $RS_{q_1}^{t_2} = \{o_1, o_3, o_5\}$, respectively, where the threshold score of query q_1, is set to 0.5, i.e., $T_s = 0.5$. In this chapter, we study *CRSK-*



(a) Boolean range



(b) Spatio-textual relevance score based range

Figure 6.1: An example of *CRSK-mo* query

mo queries by considering this general notion of relevance (for more details, see Section 6.2.2).

6.1.2 Challenges

Key challenges of solving $CRSK - mo$ queries are as follows: (i) the range threshold T_s is determined based on both spatial proximity and textual similarity; any object far from a query location with a high textual similarity score can still be the answer (and vice versa), and thus it is hard to prune

the search space, and (ii) frequent location updates may invalidate a query result, therefore, it requires continuously maintaining the the up-to-date results while minimizing the total computation cost and communication cost between clients (objects) and the server.

To address the above challenges, in this chapter, we propose a client-server based comprehensive solution for monitoring continuous range spatial keyword queries over moving spatio-textual objects. Inspired by the usefulness of *safe zone*-based approaches [4, 26, 104, 85, 117, 118] for monitoring other types of spatial queries, we also develop a safe zone based approach where each object is assigned an area (called safe zone) such that the object does not affect the result of any continuous query in the system as long as the object remains in its safe zone. The advantage is that the system does not need to recompute the results (reducing computation cost) and the object does not need to report its location to the server (reducing communication cost) as long as the object is inside its safe zone.

In addition to safe zone, we also maintain another region on the server side, which we call, *buffer region*. The buffer region reduces frequent safe zone computations and also ensures that the workload assigned to each client device is manageable. We propose a novel framework that elegantly handles frequent updates from objects while answering CRSK-mo queries. We propose a grid based in-memory data structure that enables us to efficiently process multiple long-running registered queries over a large number of moving spatio-textual objects in tandem with the efficient construction of safe zones and buffer regions. Our experimental study shows that our approach significantly outperforms the competitive *PCR* method for a wide range of parameters.

6.1.3 Contributions

Our contributions in this chapter can be summarized as follows:

- To the best of our knowledge, we are the first to study continuous monitoring of moving spatio-textual objects for multiple continuous range spatial keyword queries.
- We propose pruning rules based on spatial and textual upper bounds between queries and objects to reduce the workload by pruning a large number of irrelevant queries. Then the *safe zones* and *buffer regions* are determined by utilizing the remaining queries.
- We propose a grid based in-memory data structure that elegantly handles frequent location updates while processing multiple CRSK-mo queries.
- We conduct an extensive set of experiments to show that our proposed approach outperforms the competitive approach significantly.

The remainder of this chapter is organized as follows: Section 6.2 presents some preliminaries related to this chapter, Section 6.3 describes the proposed techniques, Section 6.5 consists of experimental evaluation. In Section 6.6 we discuss the extensions of our techniques to support the indoor space. Section 6.7 concludes the chapter.

6.2 Preliminaries

In this section, first, we discuss the limitations of existing techniques in Section 6.2.1. Then, we formulate the problem in Section 6.2.2. Finally, in Section 6.2.3, we present a descriptive discussion on our system architecture.

6.2.1 Limitations of Existing Techniques

[77, 78] suggest a technique to index moving objects trajectories. However, maintaining the index continuously for moving objects is expensive. [79, 80] introduce a strategy to index the queries instead of the objects which significantly reduces the index maintenance cost. These techniques impose a high communication and computational overhead on the server and also affect the battery life of hand-held devices due to frequent location updates and computations. To overcome these issues, [82, 79] propose an efficient and attractive technique called safe zone. In addition, [15, 85] utilize the safe zone concept to continuously evaluate moving queries. But all these techniques consider the spatial information but did not take into account the textual information. Therefore, these techniques cannot be used in continuous spatial keyword query processing over moving spatio-textual objects. However, we adopt the concept of safe zone in the context of spatio-textual queries.

[26, 104] investigate moving top-k spatial keyword queries over stationary geo-textual objects. [26] proposes a technique that uses multiplicatively weighted Voronoi diagrams. However, the proposed technique used polygons to approximate circles and thereby could not find the exact safe zone. [104] introduces a method that identify the dominant zones for objects and then used those regions to compute the safe zone for a query. Wang et al. [27] introduce a novel adaptive index called AP-tree which groups the registered queries considering their textual and spatial properties. In these studies, either they evaluate each moving query against the set of static spatio-textual objects or data static query against streaming spatial-textual data. In contrast, we evaluate each spatio-textual object against the set of queries in order to maintain an up-to-date result set with response to the movements of the objects.

Table 6.1: The summary of notations

Notation	Definition
o	a spatio-textual object
q	a continuous range spatial-keyword query
$o.\lambda(q.\lambda)$	the location of object o(query q)
$o.\psi$ $(q.\psi)$	a set of keywords for object o(query q)
$o.m$	capacity of object o
$q.T_s$	threshold score of query q
$q.\alpha$	query preference factor of query q
RS_q^t	result set of query q at timestamp t
Cl_q^o	conditional circle of object o w.r.t query q
r_q^o	radius of Cl_q^o
$BR(o)$	buffer region of object o
Cl_q^{max}	largest conditional circle of query q
r_q^{max}	radius of Cl_q^{max}
Ω	default range of buffer regions

6.2.2 Problem Statement

Let O be a set of spatio-textual objects (users) and Q be a set of facilities or POIs (queries). Each spatio-textual object $o \in O$ is defined as a pair $(o.\lambda, o.\psi)$, where $o.\lambda$ is the current point location of the user and $o.\psi$ is a set of keywords representing her preferences. Similarly, a query object $q \in Q$ is also defined as a pair $(q.\lambda, q.\psi)$, where $q.\lambda$ is the location of the facility and $o.\psi$ is a set of keywords representing its attributes in the form of a textual description. Notations frequently used in this chapter are summarized in Table 6.1.

The geo-textual relevance between an object and a query is defined in terms of both spatial proximity and textual similarity. Let $dist(q, o)$ be the spatial distance between query location $q.\lambda$ and object location $o.\lambda$, and $text(q, o)$ be the textual similarity between the two keyword sets $q.\psi$ and $o.\psi$. To convert the textual similarity to the textual distance, we use textual score as $S_t(q, o) = 1 - text(q, o)$, here a smaller value of $S_t(q, o)$ signifies a higher textual similarity between q and o. We assume our working space is normalized so that both

spatial distance score $dist(q, o)$, and textual distance score $S_t(q, o)$ lie between 0 and 1 (inclusive). Thus, geo-textual relevance score, $score(q, o)$ of o with respect to q can be expressed as follows:

$$score(q, o) = \alpha \cdot dist(q, o) + (1 - \alpha) \cdot S_t(q, o) \tag{6.1}$$

Here, α is a query parameter (user-defined) that lies between 0 and 1(exclusive) to control the preference of spatial proximity over textual similarity.

We compute the spatial proximity score $dist(q, o) = 1$ if $||q.\lambda, o.\lambda|| \geq R$ where $||q.\lambda, o.\lambda||$ is the normalized euclidean distance between q and o, and R is a system defined range. If $||q.\lambda, o.\lambda|| < R$, then $dist(q, o)$ is computed as follows,

$$dist(q, o) = \frac{||q.\lambda, o.\lambda||}{R} \tag{6.2}$$

The intuition of using R is as follows. Assume that each object has exactly three keywords. An object o's textual distance $S_t(q, o)$ will be 0 if query q contains all keywords. Its textual distance will be $1/3, 2/3$ or 1 if q contains 2, 1 or 0 of its keywords, respectively. Now, consider the example of objects in Los Angeles (the data set used in our experiments) and assume that the maximum distance between two points in the space is $100km$. Now consider two objects o_1 and o_2 such that distance between q to o_1 is 0.1 km and distance between q to o_2 is 15 km. Their spatial similarity scores (without considering R) will be their distances from q normalized in the range 0 to 1, i.e., $dist(q, o_1) = 0.1/100 = 0.001$ and $dist(q, o_2) = 15/100 = 0.15$. Now, assume that q contains 2 out of 3 keywords of o_1 (i.e., $S_t(q, o_1) = 1/3$) and q contains all three keywords of o_2 (i.e., $S_t(q, o_2) = 0$). If $\alpha = 0.5$, their total scores will be $score(q, o_1) =$

$0.5 \times 0.001 + 0.5 \times 1/3 = 0.167$ and $score(q, o_2) = 0.5 \times 0.15 + 0.5 \times 0 = 0.075$. Therefore, o_2 gets a better score although its distance from q is much larger compared to the distance of o_1. In other words, the scores are biased towards textual similarity, i.e., the objects that have better textual similarity have higher chance to be the result even if they are quite far from q.

Now, consider the same example, and assume that $R = 0.2$. In this case, $dist(q, o_1) = 0.001/0.2 = 0.005$ and $dist(q, o_2) = 0.75$ and $score(q, o_1) = 0.169$ and $score(q, o_2) = 0.375$. Note that R normalizes the spatial proximity score to reduce the bias towards the textual similarity score. In short, R was introduced to address the bias towards textual similarity. Its effect is similar to setting α to a higher value.

Note that, the textual similarity ($text$) can be computed using any information retrieval model. In this chapter, we use a function [110] similar to the weighted *Jaccard coefficient*, described as follows:

$$text(q, o) = \frac{\sum_{t \in q.\psi \cap o.\psi} w(t)}{w(o.\psi) = \sum_{t \in o.\psi} w(t)} \tag{6.3}$$

where, $w(t)$ denotes the weight of keyword t, computed by obtaining the inverted document frequency (idf). And $w(o.\psi)$ indicates the weighted sum of object keywords (i.e., $o.\psi$).

Definition 6.1 ***Range Spatial Keyword Query (RSKQ)***
Let O be a set of spatio-textual objects and, q be a range spatial keyword query, $q = \{\lambda, \psi, \alpha, T_s\}$ where λ is the query location, ψ is the set of keywords, α the query preference factor between spatial proximity and keyword set similarity, and T_s is the range threshold score combining both spatial and textual

factors. The query q returns a set of objects, $RS_q \subseteq O$, whose geo-textual relevance scores are less than or equal to the given threshold score T_s, i.e, $\forall o^* \in RS_q, score(q, o^*) \leq T_s$.

Definition 6.2 ***Continuous Range Spatial Keyword Query on Moving Objects (CRSK-mo)***

Let O be a set of moving *spatio-textual objects, and Q be a set of long running static range spatial keyword queries, for each $q \in Q$, the continuous range spatial keyword query over moving objects (CRSK-mo) finds a set $RS_q^t \subseteq O$ of objects for* every time instance *t, where* $\forall o^* \in RS_q^t, score(q, o^*) \leq T_s$.

6.2.3 Client-Server Model:

We utilize the client-server paradigm, in which we have two types of client objects: facilities (static) and users (moving). The facilities (e.g., restaurants, shops, etc.) are static clients who issue queries to the server. The users are moving clients who use their GPS-enabled mobile phones to continuously track their respective locations, and send updates to the server. The latter type of clients is referred as spatio-textual moving objects in this chapter. Figure 6.2 shows the schematic diagram of our system model where clients send their updates and issue queries to the central server, and the server is responsible for maintaining moving object and processing the queries. Finally, the server returns the result sets to the clients.

The current location of object o is represented using x-y coordinates, i.e., $(o.\lambda.x, o.\lambda.y)$. Initially, an object sends its location and preferences (keywords) as a tuple $(o.\lambda, o.\psi)$. Since objects frequently change their positions (i.e., mov-

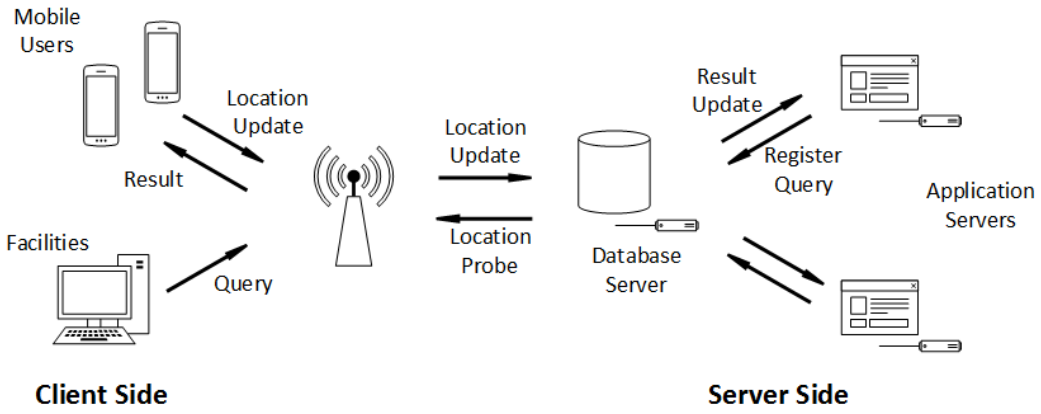


Figure 6.2: The system architecture

ing), an object o sends its location update to the server as $< oID, \lambda.x_{cur}, \lambda.y_{cur} >$, where (x_{cur}, y_{cur}) is the current location of object o.

6.3 Solution Overview

In this section, we present a comprehensive solution for monitoring continuous range spatial keyword queries over moving objects. Processing continuous queries over moving objects is more challenging since a slight movement of an object may invalidate a query result. Thus, it requires monitoring the locations of the objects and maintaining the results continuously as the objects move. To address this challenge, we utilize the concept of *safe zones* [83, 84, 82, 79]. The safe zone of an object is an area such that as long as the object remains inside this area, it does not affect the result of any query. Hence, the object does not need to send location updates to the server unless it leaves the safe zone. Thus, the safe zone based approach reduces both the query processing cost and the communication cost between clients and the server.

In CRSK-mo query processing, the query range is determined based on both spatial proximity and textual similarity; any object far from a query location

with a high textual similarity score can still be an answer for the query (and vice verse). The key idea of our approach is to exploit the spatial and textual upper bounds between queries and objects to form safe zones for each object. We also introduce the concept of buffer regions, which is maintained in the server side to avoid frequent re-computations of safe zones. Moreover, we utilize these bounds to quickly prune queries through efficient in-memory data structures. We describe our solution in the following subsections.

Section 6.3.1 presents the concept of safe zones that form the basis of our algorithm. Section 6.3.2 presents the pruning rules based on spatial proximity and textual similarity that are used by our algorithm to prune the search space.

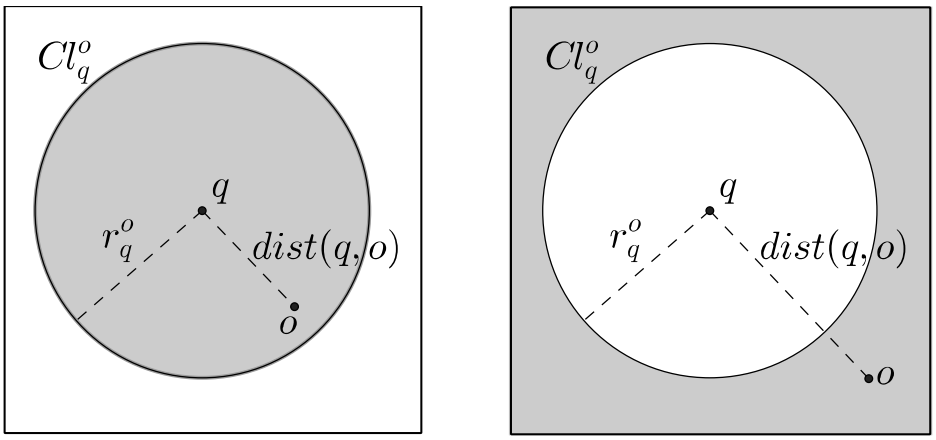
6.3.1 Safe zone of an object

In a range spatial keyword query, an object o is a result for a given query q when the spatio-textual relevance score of the object is less than or equal to the given query threshold score, i.e., $score(q, o) \leq T_s$. To continuously monitor an object for a registered query, we need to essentially monitor the corresponding inequality over the time. Hence, we have the following lemma formalizing it.

Lemma 6.1 *An object $o \in O$ is a result of query q (i.e., $o \in RS_q$) iff* $dist(q, o) \leq \frac{T_s}{\alpha} - \frac{(1-\alpha)}{\alpha} \cdot S_t(q, o)$.

Proof Let q be a range spatial keyword query and an object o be one of the results of the query q, i.e., $o \in RS_q$. To satisfy the query condition, the object o must follow the condition, $score(q, o) \leq T_s$ as depicted in the Definition 6.1. Thus, we can rewrite Equation 6.1 as follows: $\alpha \cdot dist(q, o) + (1-\alpha) \cdot S_t(q, o) \leq T_s$. Hence, $dist(q, o) \leq \frac{T_s}{\alpha} - \frac{(1-\alpha)}{\alpha} \cdot S_t(q, o)$.

Based on the above lemma, we define a circle called *conditional circle*, Cl_q^o, centred at the query location $q.\lambda$ with the radius of r_q^o, where the radius is



(a) (b)

Figure 6.3: Example of conditional areas

defined as follows:

$$r_q^o = \frac{T_s}{\alpha} - \frac{(1-\alpha)}{\alpha} \cdot S_t(q,o) \tag{6.4}$$

Intuitively, a conditional circle, Cl_q^o, is *a spatial area* such that object o does not affect the result of the query q as long as o does not enter or leave the area. We identify this area as the *conditional area*. Figure 6.3(a) shows an example of the conditional area (shaded in gray) for object o when the object resides inside (i.e $dist(q,o) \leq r_q^o$) the conditional circle, i.e., Cl_q^o. In this case o remains as a result of q as long as it resides inside the cirlce. Otherwise, the object o is outside the Cl_q^o, then $o \notin RS_q$ and the conditional area is the shaded area as shown in Figure 6.3(b).

Intuitively, the safe zone of an object involving multiple queries is constructed by taking the intersection of the conditional areas of the object with respect to all the queries. For example, Figure 6.4 shows conditional circles of object o_1 and o_2 for queries q_1,q_2,q_3 and q_4. Since o_1 only lies inside the $Cl_{q_2}^{o_1}$ and $Cl_{q_3}^{o_1}$, the safe zone of the object o_1 is the shaded area as shown in Figure 6.4(a). As long as the object o_1 resides inside this area, it does not affect

the results of any query. The object o_1 can determine whether it is inside the safe zone by checking whether it is inside the two conditional circles $Cl_{q_2}^{o_1}$ and $Cl_{q_3}^{o_1}$.

The number of conditional circle boundaries that an object can monitor entirely depends on the computational capability of the object (i.e., client device). An object with low computational capability may result an overwhelming workload on the processor and short battery life, if the safe zone assigned to that object involves a large number of conditional circles. For example, assume that both objects o_1 and o_2 have the same set of keywords, i.e., conditional circles for each query is identical for both objects because the textual similarity is same. Figure 6.4(b) shows the safe zone of object o_2 shaded in gray, which is outside of all the conditional circles. In such a scenario, the object may need to monitor a large number of conditional circles, which is computationally expensive.

To address this problem, we introduce a concept called *Buffer Region (BR)*, to support clients' mobile devices with heterogeneous computational capabilities. Based on the computational capability of each registered object, the server assigns a value called *capacity* (denoted by m) which is the maximum number of conditional circles that particular object can monitor at a time. Thus, the capacity of the object is used to bound the number of conditional circles involved in constructing the safe zone of that object. Thereby, each device is assigned with a reasonable computational workload.

Definition 6.3 Buffer Region (BR)

Let $o \in O$ be a spatio-textual object, where $o = \{\lambda, \psi, m\}$. The Buffer region is a circle centered at $o.\lambda$ and the radius is the Euclidean distance from $o.\lambda$ to the $m - 1^{th}$ nearest conditional circle. We denote the buffer region of the object o by $BR(o)$.

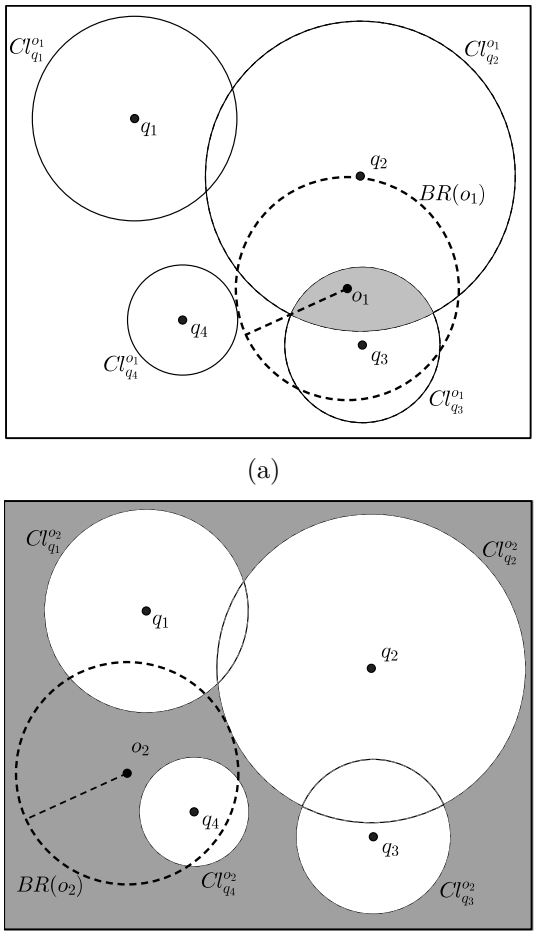


Figure 6.4: Buffer regions for (a) object o_1, (b) object o_2

The buffer region of an object includes nearest $m-1$ conditional circles ensuring that the number of conditional circles involved in constructing the safe zone of the object does not exceed the capacity of the object. Figure 6.4 shows the buffer regions (i.e., dotted circle) and the safe zones (i.e., shaded area) of object o_1 and o_2. After the buffer region is constructed, it is stored in the server and the safe zone is sent to the client device of the particular object. For example, assume o_1 is an object in O, where $o_1.m = 4$. Then the buffer region of object o_1 involves three (i.e., $m-1$) conditional circles (see Figure 6.4(a)). Thus, o_1 will monitor four circle boundaries including the buffer region boundary. Hence, the radius of the buffer region is $dist(o_1, Cl_{q4}^{o_1})$. Figure 6.4(b) shows the buffer region for object o_2 assuming its capacity is also four. Thus, it reduces the number of conditional circles that object o_2 has to monitor to check whether it is inside the safe zone. Note that the buffer region concept reduces the frequent safe zone computations and ensures the workload assigned to each client device is manageable.

6.3.2 Pruning Rules

Processing CRSK-mo queries involves computing the results of the queries and constructing the buffer regions for each object to reduce the computational and communication overhead. Naively, for each object, we can compute conditional circles for all the queries to determine the affected queries (i.e., the queries for which the object is a result) and also to construct the buffer regions. Since the number of objects and queries are large in numbers, this naive approach is highly inefficient. To avoid this limitation, in this section, we introduce two simple pruning rules based on the bounds derived from spatial proximity and textual similarity between an object and a query. Using these pruning rules we filter a large number of irrelevant queries and obtain a set of candidate

queries. Then we compute conditional circles for these candidate queries to identify the queries for which the object is a result.

According to Equation 6.4, the radius of the conditional circle depends on threshold score, textual relevance, and preference parameter(α). Since the textual relevance is the only value that can vary from one object to another object, we can obtain an upper bound for the radius of the conditional circle when we set the textual relevance as zero (i.e., $S_t(q, o) = 0$). Thus, we define Cl_q^{max} as the *largest conditional circle* of query q, where the radius r_q^{max} can be defined as follows,

$$r_q^{max} = \frac{T_s}{\alpha} \tag{6.5}$$

Lemma 6.2 ***(Pruning Rule 1)*** *Let an object $o \in O$ be outside the Cl_q^{max} of query q, then object o cannot be a result for the query q.*

Proof Let an object o be a result of query q. According to Lemma 6.1, $dist(q, o) \leq r_q^o$. Hence, $r_q^o \leq r_q^{max}$, and $dist(q, o) \leq r_q^{max}$. It concludes that object o is not a result of the query if $dist(q, o) > r_q^{max}$.

In line with Pruning Rule 1, if the object o is outside the Cl_q^{max}, then the query q can be pruned. Otherwise, the conditional circle of object o for query q, i.e., Cl_q^o is computed to verify whether the object is a result of the query (according to Lemma 6.1). Figure 6.5 shows an example for Pruning Rule 1. The solid circles depict the largest conditional circles of queries q_1,q_2 and q_3 while dotted circles depict the conditional circles of object o_1 for each query. Since object o_1 is inside the $Cl_{q_1}^{max}$ and $Cl_{q_2}^{max}$, queries q_1 and q_2 are identified as candidates while query q_3 is filtered out. Since o_1 is inside the $Cl_{q_1}^{o_1}$, object o_1 can only be a result for q_1.

However, Pruning Rule 1 may include a large number of candidate queries since it only considers the spatial proximity. So we present our next pruning rule that exploits textual relevance to filter the irrelevant queries. Next, we determine an upper bound (denoted by $maxT$) for the textual similarity between an object and a query as follows.

From the scoring function (Equation 6.1), if an object o is a result, then $\alpha \cdot dist(q,o) + (1-\alpha) \cdot S_t(q,o) \leq T_s$. Hence, when the $dist(q,o) = 0$, Equation 6.1 can be rewritten as follows.

$$(1-\alpha) \cdot S_t(q,o) \leq T_s$$
$$S_t(q,o) \leq \frac{T_s}{(1-\alpha)}$$

Thus, the $maxT_q$ can be expressed as follows.

$$maxT_q = \frac{T_s}{(1-\alpha)} \tag{6.6}$$

Lemma 6.3 ***(Pruning Rule 2)*** *An object o has potential of becoming a result of query q if $S_t(q,o) \leq maxT_q$.*

Proof Let o be an object with $S_t(q,o) > maxT_q$. By Equation 6.6, $S_t(q,o) > \frac{T_s}{(1-\alpha)}$. Hence, we have $\frac{T_s}{\alpha} - \frac{(1-\alpha)}{\alpha} \cdot S_t(q,o) < 0$. By Equation 6.4, $r_q^o < 0$ concludes that object o ca never be a result of query q.

If an object does not satisfy Pruning Rule 2 for a query, then that object can never be a result for the query as the conditional circle of that object does not exist(i.e., $r_q^o < 0$). Thus, we filter those queries in order to reduce the search space. Consider the previous example (see Figure 6.5), if $S_t(q_2, o_1) > maxT_{q2}$ then q_2 will be filtered out even though the Cl_{q2}^{max} overlaps with object o_1.

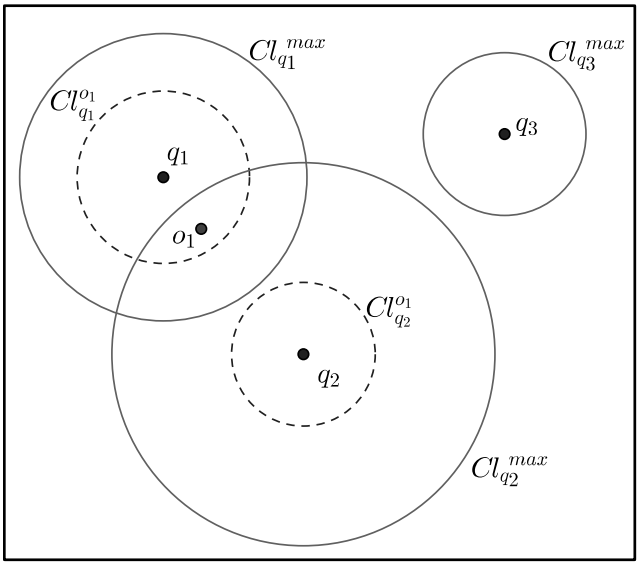


Figure 6.5: An example for pruning rules

6.4 Algorithm

In this section, we discuss our algorithm for processing CRSK-mo queries. We first present, the server-side query processing framework, and then we propose a filter-verification algorithm that constructs buffer regions for each object while computing the result sets for each query. After that, we discuss continuous monitoring of range spatial keyword queries with respect to location updates of the objects. Finally, we present client-side processing of the system.

6.4.1 The Framework

We use a *gird-based index structure* to index the CRSK-mo queries. We prefer grid-based index over the other index structures like an R-tree as it supports frequent location updates and also it is usually preferred in continuous query processing [86, 119, 87]. In our index, the data space is partitioned into $2^n \times 2^n$ grid cells (where $n \geq 0$) as shown in Figure 6.6. To access a particular cell quickly, we consider the grid as a conceptual tree as used in [4]. The root of

the conceptual tree is a rectangle that covers the whole work space (i.e., all the cells). The root cell is divided into four equal grid cells that represent the next level of the tree. The process continues until each entry of the leaf level represents one grid cell.

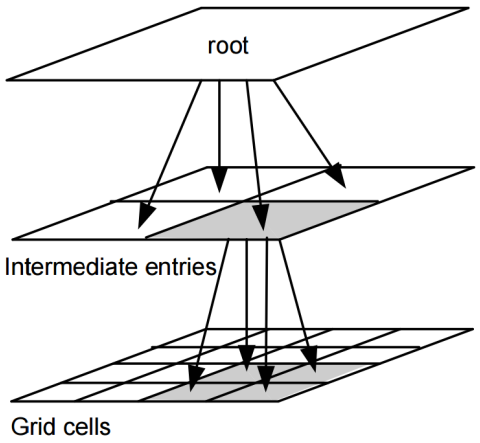


Figure 6.6: Conceptual grid-tree of a 4×4 grid [4]

Since the grid-tree is a conceptual visualization of the grid, the root entry and intermediate entries do not exist physically. So that the information is stored only in leaf level grid cells. Hence, in each grid cell, we store all the queries whose Cl_q^{max} circles overlap with the particular cell. So that when an object lies inside the cell, we can filter out all the other queries according to Pruning Rule 1. To efficiently access the queries based on keywords, instead of using a flat list we use an *inverted list* (i.e., $iQList$) to store these overlapping queries. Figure 6.7 shows our grid based index structure that consist of inverted lists at each grid cell. In the inverted index, for each keyword k_i, we store the queries whose description contains k_i. Accordingly, each keyword contains a posting (query) list ordered by the query id. Thus, we access the inverted list by using *document at a time* access method. Thereby, we consider only the queries contains at least one matching keyword with the objects. Moreover, by using the inverted index and Pruning Rule 2, we obtain the candidate set

more efficiently.

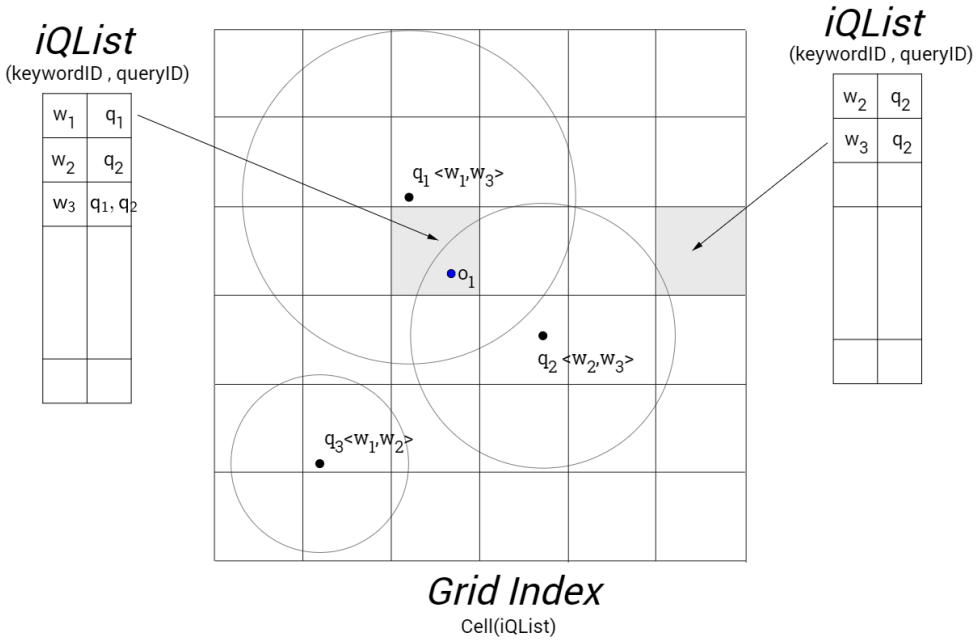


Figure 6.7: Example of the index structure

6.4.2 CRSK-mo Processing

We assume that all registered queries are indexed using our grid-based index structure, and objects arrive into the system in a stream-like fashion. As soon as an object arrives, it is immediately evaluated by our algorithm to see whether it can affect any query result. At the same time, our algorithm determines the buffer region (BR) for the object in tandem with the query evaluation. The buffer region is stored in the server and the safe zone is sent to the device of the corresponding object. Finally, query results are reported to the respective query client.

Our approach consists of two phases: filtering phase and verification phase. In the filtering phase, all the queries whose Cl_q^{max} do not overlap with the object

location (Pruning Rule 1) and the queries whose textual relevance is greater than its $maxT_q$ (Pruning Rule 2) are filtered out. Then, in the verification phase, conditional circles for each candidate query are computed and the result sets of candidate queries are updated. Finally, the buffer region and the safe zone of the object are constructed.

Algorithm 11 shows the pseudocode of our algorithm. The algorithm takes an object o, and grid-based index G as input, and updates the result sets of the queries and returns the buffer region and safe zone of the object o. We start traversing the conceptual grid-tree from the root. The root entry is first inserted into the priority queue, Q_p. The elements in the priority queue are maintained in increasing order of their minimum Euclidean distance from the object. If the dequeued element is an intermediate cell and satisfies a system defined default range denoted by Ω (later we explain the intuition of this value), then we insert its children into the queue, where $mindist(chlidCell, o)$ is the key (Lines 5-7). If the dequeued element is a cell, we use the inverted list $iQList$ of the cell to select a candidate list of queries $cQList$, and for each $q \in cQList$, we compute the conditional circle Cl_q^o, and finally, the object o is inserted into the result set for the query q if the object falls inside the conditional circle Cl_q^o. Note that, when we process the inverted list of each cell, we record the queries already processed in order to avoid the redundant access.

To compute the buffer region in tandem with the CRSK-mo query processing, we continue inserting the conditional circle into the priority queue, where $mindist(Cl_q^o, o)$ is used as the key. If the dequeued item is a Cl_q^o then it is added to the list maintaining conditional circles for computing the buffer region. When this list size becomes $m - 1$, we stop the process since we have sufficient Cl_q^os to construct the buffer region of object o, $BR(o)$ (Lines 22-24).

Algorithm 11: RSKQInit(o)

Data: Object o, Grid-index G
Result: RS_q for all $q \in Q$, $BR(o)$

```
1  Q_p.Enqueue(G.root,0)
2  while Q_p NOT Empty do
3      element ← Q_p.Dequeue()
4      if element is an intermediate cell then
5          foreach childCell ∈ element do
6              if mindist(childCell, o) ≤ Ω then
7                  Q_p.Enqueue(childCell, mindist(childCell, o))
8              end
9          end
10     else if element is a cell then
11         cQList = getCandidateList(o, iQList)
12         foreach q ∈ cQList do
13             if S_t(q, o) ≤ maxT_q then
14                 Compute Cl_q^o ;          // compute conditional circle
15                 Update RS_q ;                    // update query answer
16                 if mindist(Cl_q^o, o) ≤ Ω then
17                     Q_p.Enqueue(q, mindist(Cl_q^o, o))
18                 end
19             end
20         end
21     else
22         o.CLList.add(element, element.key) ;   // add to candidate set
23         if sizeof(o.CLList) = m − 1 then
24             break;
25         end
26     end
27 end
28 BR(o) ← computeBR(o.CLList) ;          // computing buffer region
```

Note that, it may happen that an object has less than $m-1$ nearby queries so that it is required to traverse a wider area to compute its buffer region. In such a scenario, we take a system defined default range for the buffer region, denoted by Ω, that bounds the traversal area. From the conditional circle list, $CLList$ of object o, we compute the buffer region of the object, $BR(o)$ in Line 28. Thus, the safe zone of the object o is also determined while the buffer region is generated.

6.4.3 Continuous Monitoring

Since the objects may frequently update their positions, we need to update the results of all the queries. In this section, we discuss our approach for handling frequent location updates of moving objects for processing CRSK-mo queries. When an object o sends its location to the server, the server performs the following steps to update the results (see Algorithm 12). First, the server checks whether the new location is inside the current $BR(o)$, if it is true, then it checks against each conditional circle that forms the $BR(o)$ to identify which queries have been affected by this location update. Then the affected queries are updated accordingly. If the new reported location is outside the current $BR(o)$, a new buffer region needs to be computed using Algorithm 11.

6.4.4 Client Side

In our system, we assume that the client tracks its own location and nearby conditional circles. Since the client has a limited computational capability and wants to be a part of a limited number of nearby conditional circles, each client is assigned with a capacity (i.e m) to limit the number of conditional circles that involve in constructing the safe zone. The client initially sends its current

Algorithm 12: Continuous monitoring

Data: Location Update $< o, \lambda_{new}, \lambda_{old} >$
Result: Update RS_q for all $q \in Q$, $BR(o)$
`// determining the affected queries`
if *Object* $o.\lambda_{new}$ *is inside* $BR(o)$ **then**
  foreach $Cl_q^o \in o.CLList$ **do**
    `// Update the query results`
    if $o.\lambda_{old}$ *is outside* Cl_q^o ***AND*** $o.\lambda_{new}$ *is inside* Cl_q^o **then**
      Insert o into RS_q
    else if $o.\lambda_{old}$ *is inside* Cl_q^o ***AND*** $o.\lambda_{new}$ *is outside* Cl_q^o **then**
      Delete o from RS_q
  end
else
  $BR(o) \leftarrow computeBR()$
end

point location and a set of keywords to the server. Then the server evaluates the object against all registered queries, and sends the safe zone to the object. Subsequently , the client sends its location update to the server when it leaves the current safe zone and then the server sends back a new safe zone.

6.5 Experimental Evaluation

In this section, we evaluate the performance of our algorithm (Our) by comparing with two competitive algorithms. Section 6.5.1 explains the PCR approach. Section 6.5.2 introduces the parameters and the settings we used in our experiments while Section 6.5.3 describes how the default parameters were determined. Section 6.5.4 compares our algorithm with a spatial filtering based algorithm . Section 6.5.5 presents a detailed discussion on empirical studies.

6.5.1 Pre-Circular Range (PCR) Approach

The straightforward approach for processing CSRK-mo queries involves evaluating each incoming object updates against all the registered queries, which is

very expensive in terms of computational and communication overhead. Thus, we develop a competitive approach called Pre-Circular Range (PCR) to compare with our technique. In the PCR approach, for each object, we first identify a set of queries that can be affected by the movement of the object for a certain period of future time. We set a circular area, $Cir(o)$ around the object where the object can belong in a defined period of time. Thus, we first identify a set of queries whose results can be affected by the movement of the object within this circular area. In general, there can be four categories of queries: (i) a query whose Cl_q^o is completely inside the $Cir(o)$, (ii) a query whose Cl_q^o partially overlaps with $Cir(o)$, (iii) a query whose Cl_q^o is completely outside $Cir(o)$, and (iv) a query whose Cl_q^o completely contains $Cir(o)$. Naturally, the movement of object o within $Cir(o)$ only affects the queries, say $AQ(o)$, that fall under category (i) and (ii), as the boundaries of these Cl_q^os can be crossed by the object. Then we select the nearest $m-1$ queries from $AQ(o)$ to construct the $BR(o)$. If the object goes outside $BR(o)$, we need to assign a new $Cir(o)$ and repeat the above procedure. The communication between the client and the server remains similar to our approach where an object sends a location update to the server as it leaves the safe zones and so on.

6.5.2 Experiment Settings

The experiments were conducted on a dataset which was generated as follows. We used a real-world dataset (from Yelp) that contains check-in data in Los Angeles to extract the POIs. The keywords for POIs were collected from the descriptions of the relevant user check-ins. Thus, the trajectories of the moving objects were generated using the brinkhoff data generator [120] based on the road network of the Los Angeles. Then we selected a set of POIs for each object by taking the nearest POI to the object trajectory at different

timestamps assuming the users checked-in to those places. Finally, we obtained the keywords for each object (user) by randomly selecting keywords from the checked-in POIs.

We have varied a wide range of parameters to test the supremacy of our approach over the PCR approach in a wide variety of real world settings. The details about the parameter values are given in Table 6.2. All the experiments were conducted in an Intel processor of 3.30GHz and 4GB of RAM, running Linux Ubuntu. We used C++ to implement all the algorithms and used in-memory setting by adopting the grid index structure since continuous result computation is essential.

We have measured the query processing time on the server side in two different metric: *initial time* and the *continuous monitoring time*. The initial time is the time spent to compute results for all the queries and construct safe zones for all the objects for the first time. Since all the queries are continuously monitored for 100 timestamps, in each timestamp server needs to maintain an up-to-date result set for all the queries as it receives location updates. In which server updates the query results, compute new buffer regions and send new safe zones to the particular objects. The continuous monitoring time sums up the time consumed by the server to maintain an up-to-date result set for the duration of 100 timestamps. In Section 6.5.5, we use the term *continuous time* to represent the *continuous monitoring time* for brevity.

6.5.3 Default Parameters

In this section, we describe how the default parameters were determined to obtain the best performance of the proposed framework for all simulations.

Table 6.2: The parameters used for experiments

Parameter	**Default**	**Range**
Threshold score	0.5	0.1 - 0.9
Preference parameter	0.5	0.1 - 0.9
Keywords per object	5	2 - 10
Keywords per query	15	10 - 30
Number of queries	10K	5K - 20K
Number of objects	100K	50K - 200K
Speed	medium	slow, medium, fast

Grid size

We conducted experiments to study the effect of the grid size. Figure 6.8 shows the effect of grid size where we change the grid size from 1×1 to 32×32 and report the continuous time. The performance degrades if grid size is too small or too large. This is because if grid cells are too large then the number of queries in each cell (and the size of inverted index) increases resulting in a poor performance. On the other hand, when the grid cells are too small, the algorithm needs to access more cells of the grid to compute the safe zone resulting in a higher computation time. Based on these experiments, we chose 4×4 as our default grid size in the experiments.

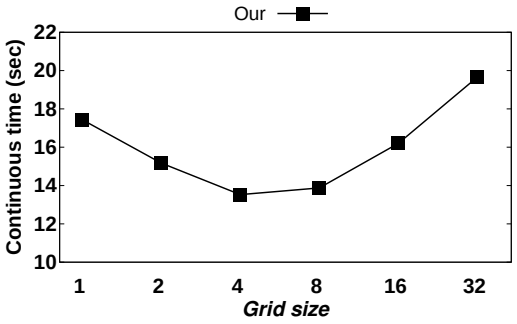
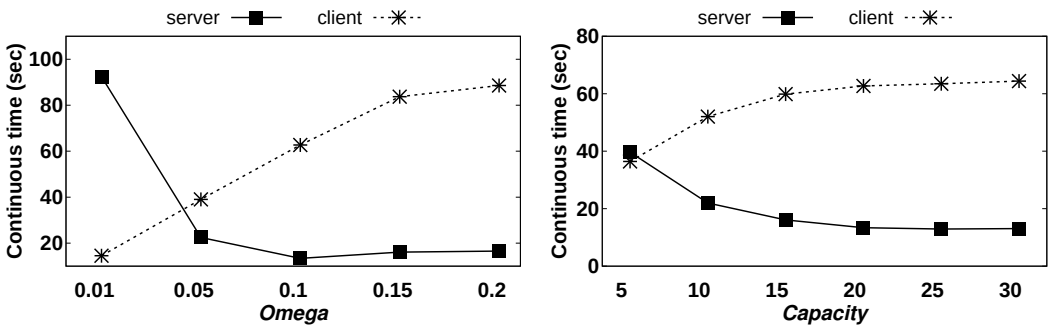


Figure 6.8: Effect of grid size

Omega value

In Figure 6.9(a), we conducted experiments by varying the default range (i.e., Ω) and study its effect on the total cost at the server side and the total cost at the client side. Figure 6.9(a) shows that the total cost at server side reduces as Ω increases. This is because, as Ω increases, the buffer region size increases which results in requiring to recompute the buffer regions fewer times. In contrast, the computation cost at the client side increases as Ω increases because the area that the client device needs to monitor becomes larger. In our experiments, the default value of Ω is set to 0.1 with an aim to minimize the total cost at the server side.



(a) (b)

Figure 6.9: Varying Ω and m

Capacity

Recall that the capacity of an object (i.e., m) is decided based on the computational power of each client device (i.e., object) because the system consists of objects with heterogeneous computational capabilities. In Figure 6.9(b), we study the effect of capacity on the total computation cost on the server and the total computation cost on all client devices for all 100 timestamps.

Figure 6.9(b) shows that the total cost on server side is reduced as the capacity increases. This is because the size of buffer region increases as the

capacity increases and, as a result, the server needs to update the buffer regions fewer times. In contrast, the total cost on client devices increases as the capacity increases. This is because, to check whether a client is inside its safe zone or not, it needs to check its location against m circles and this cost increases as m increases. Therefore, there is a trade off in choosing a suitable value of m. In this chapter, we choose $m = 20$ to optimize the total cost at the server side. In real world scenarios, the client devices may be asked for their preferred values of m based on their computational capabilities.

6.5.4 Comparison with Spatial Filtering based Approaches

Even though a simple spatial filtering based approach (e.g., range queries) may not work, we designed another competitor that first filters based on a particular range ρ and then retrieves the results among the filtered queries. The range ρ is set assuming the maximum possible textual similarity for each object, i.e., ρ is set such that an object o which has distance greater than ρ from q cannot be an answer even if it has maximum textual relevance. We call this approach spatial filtering (SF). This approach first applies spatial filtering based on distance ρ and then processes the candidates within the range to determine if they are the results or not.

Figure 6.10 compares the performance of our approach, PCR and SF methods for different number of queries. Our algorithms and PCR both outperform SF approach and scale much better with the increase in the number of queries. The performance of SF severely deteriorates as the number of queries increases mainly because more queries are found in the filtering range ρ and require verification. Since the performance of SF is comparatively much worse than PCR,

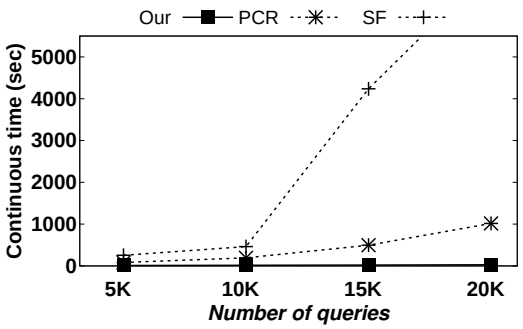


Figure 6.10: The performance of our approach, PCR and SF

we compare our algorithm only with PCR in the forthcoming experiments.

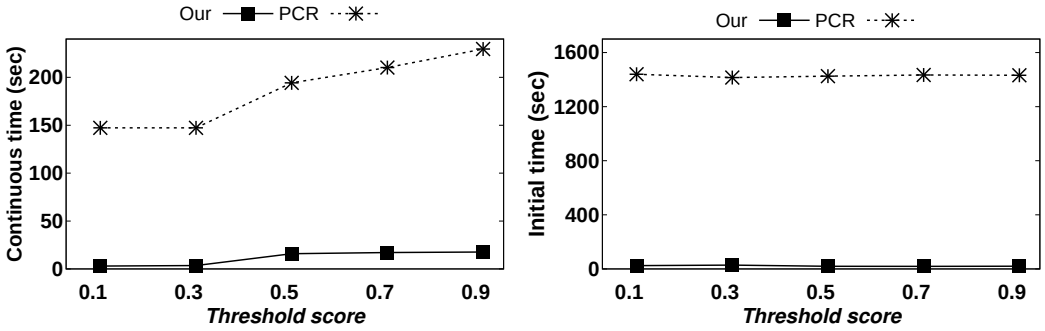
6.5.5 Performance Evaluation

In this section, we present the experimental results of proposed algorithm and compare our approach with the pre-circular range (PCR) approach by varying different parameters.

Varying query parameters

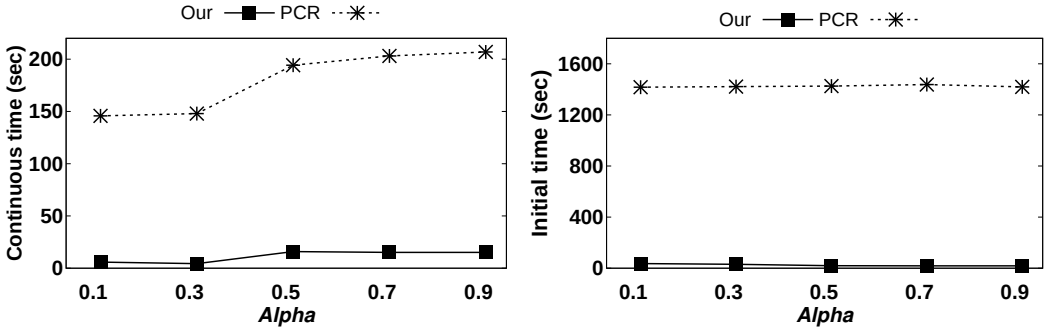
In this experiment, we evaluate the performance of our algorithm by varying the threshold score from 0.1 to 0.9 . As the Figure 6.11 shows, our algorithm significantly outperforms PCR approach in terms of continuous time and initial time due to the efficient filtering techniques. With the increase of threshold score value, the continuous time shows an increasing trend for both algorithms as higher threshold scores incur large result sets.

Moreover, we varied the value of alpha from 0.1 to 0.9 . Figure 6.12 shows results. It can be clearly seen that our algorithm performs better compared to PCR when the alpha is increased. Furthermore, the continuous time of our algorithm is approximately 12 times faster than PCR while the initial time of our algorithm outperforms PCR in two orders of magnitude for both settings.



(a) (b)

Figure 6.11: Varying threshold score

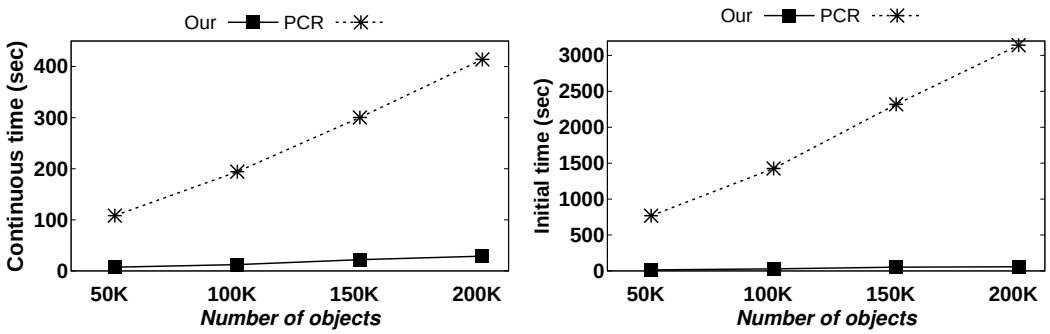


(a) (b)

Figure 6.12: Varying alpha

Varying number of objects

We evaluate the performance with respect to the number of objects (users). The number of objects scales from 50K to 200K. Figure 6.13 shows the performance of both algorithms decrease as the number of objects is increased. But our algorithm performs much better in all cases due to the efficient pruning techniques. PCR performs 10 times slower than our algorithm when the both algorithms run with 200K objects. Moreover, the initial time of our algorithm is much smaller compared to PCR.

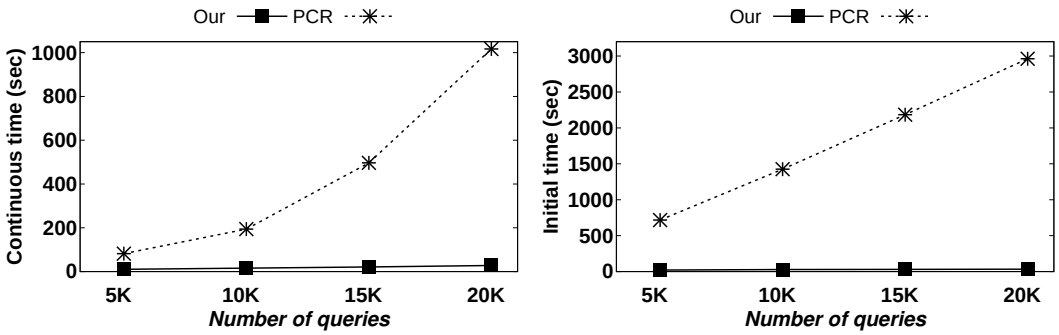


(a) (b)

Figure 6.13: Varying the number of objects

Varying number of queries

Figure 6.14 studies the performance of our algorithm with respect to the number of queries. We scaled the number of queries from 5K to 20K. Obviously, the continuous time of the both algorithms increases as the the number of queries increases. Note that, our algorithms performs much better compared to PCR due to the effective pruning techniques. Moreover, our algorithm performs 40 times faster when the number of queries is 20K and also the initial time of our algorithm outperforms PCR in two orders of magnitude.



(a) (b)

Figure 6.14: Varying the number of queries

Varying timestamps

To evaluate the efficiency of the proposed algorithm, we varied the monitoring time interval from 50 to 200 timestamps. Figure 6.15 shows the effectiveness of our algorithm with respect to PCR. PCR perform much worse as the time interval is increased since number of times that the buffer regions are generated increases.

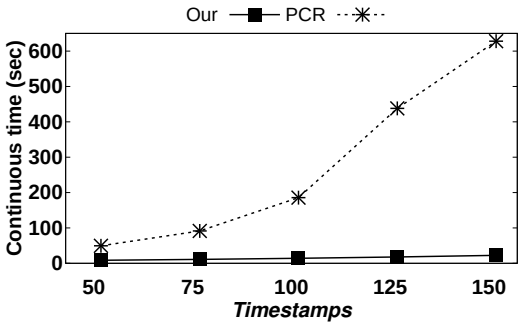
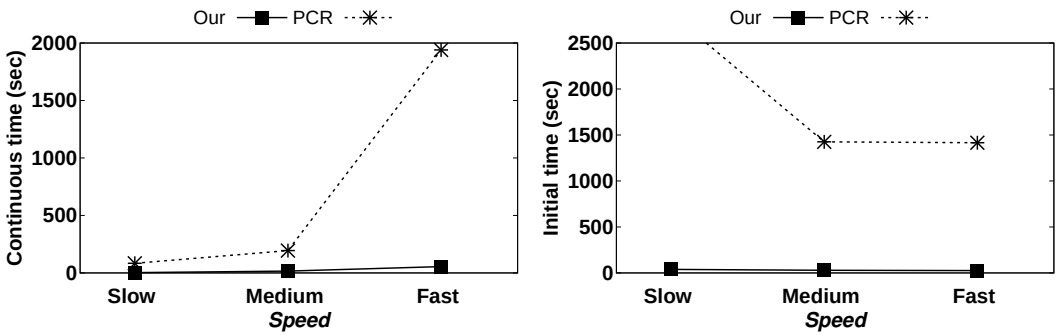


Figure 6.15: Varying timestamps

Varying speed

In this experiment, we study the effect of the speed of the objects on the performance of our algorithm. Figure 6.16 shows an increasing trend in continuous times for both algorithms. This happens because the probability of an object leaving its buffer region is proportional to the moving speed of the object. Thus, when the speed increases, the performance starts to decrease as the number of times the server regenerates the buffer regions increases. The performance of PCR decreases dramatically since its computation cost of regenerating a buffer region is really high. Moreover, our algorithm performs 40 times faster than PCR when the objects move fast.

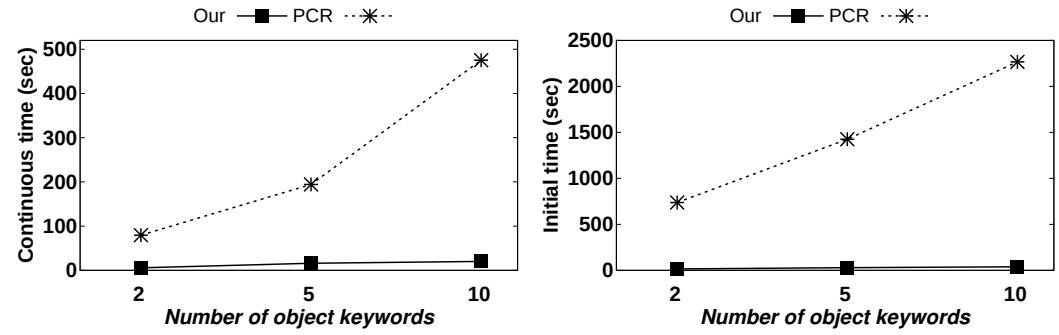


(a) (b)

Figure 6.16: Varying the speed

Varying number of keywords

Figure 6.17 and Figure 6.18 illustrate the effect of number of keywords in objects and queries respectively. Both algorithms present an increasing trend as the number of keyword is increased. Our algorithm performs much better compared to PCR since our algorithm uses inverted indexes to filter the queries based on textual relevance. Moreover, our algorithm performs 30 times faster than PCR when each query has 30 keywords.

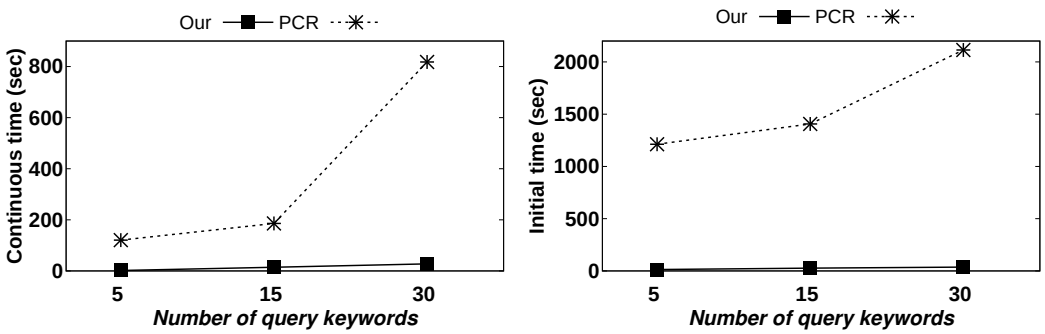


(a) (b)

Figure 6.17: Varying the number of object keywords

Effectiveness of safe zones

In this experiment, we illustrate the effect of safe zones on communication cost. In Figure 6.19, we study the effect of capacity (which affects the safe zone



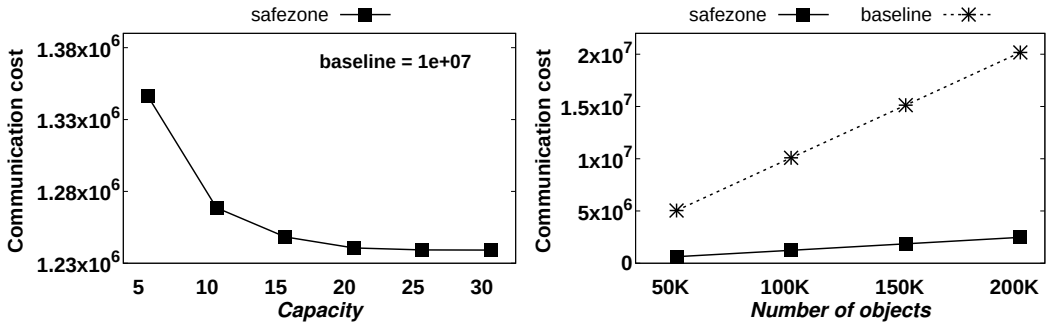
(a) (b)

Figure 6.18: Varying the number of query keywords

size) and the effect of total number of objects (users) on total communication cost. The baseline approach requires every object to send its location at every timestamp. In contrast, the safe zone based approaches (our and PCR) require the objects sending their locations only when they leave their respective safe zones. Since PCR is designed such that it always assigns the same safe zone as our approach, it has the same total communication cost as our approach. Figure 6.19 shows that safe zones significantly reduce the total communication cost. In Figure 6.19(a), the total communication cost reduces as the capacity increases because the safe zone size increases with the increase in capacity.

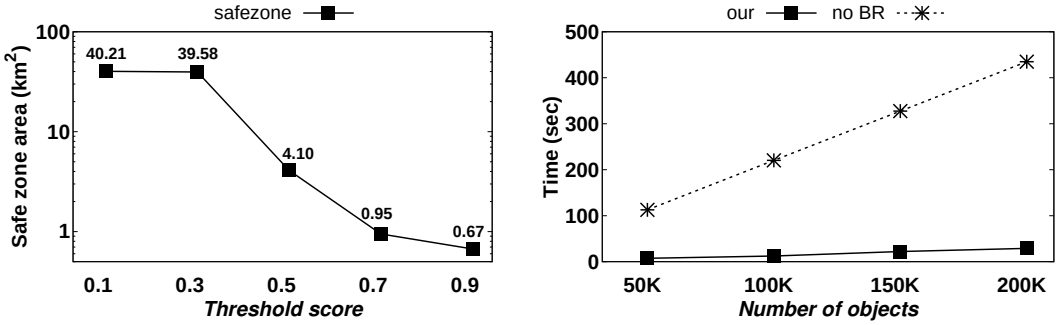
Moreover, Figure 6.20(a) shows the *average* size of safe zones for different thresholds. Note that we designed our competitor PCR such that it assigns the same sized safe zone as our approach and then retrieves queries to guarantee that the results are unaffected as long as the objects remain in their respective safe zones. Therefore, the safe zone size for both approaches is the same. Figure 6.20(a) shows that the size of safe zones reduces as the threshold increases. This is mainly because, as the threshold increases, the object is an answer for more queries which results in a reduced safe zone size. Nevertheless, even for large thresholds, the safe zone is reasonably large ($0.67km^2$ – roughly $800m \times 800m$) which is critical for its effectiveness, e.g., the safe zone based

approach is effective when an object stays in it for longer.



(a) Varying capacity (b) Varying number of objects

Figure 6.19: Communication cost



(a) Average area of safe zones (b) Effectiveness of buffer region

Figure 6.20: Experiments on safe zones and buffer regions

Effectiveness of buffer regions

In order to show the effect of buffer regions on client workload, we evaluate the effect of buffer regions on the total computation cost at the client devices. Specifically, we compare our approach that uses the buffer region with a version (denoted as "No BR") that does not use buffer regions (i.e., default range Ω and the capacity m are both set to infinity). In Figure 6.20(b), we study the effect of total number of users (client devices) on the total computation cost on *all clients for all* 100 *timestamps*. Figure 6.20(b) shows that the buffer region reduces the client side computation cost by up to four times.

6.6 Extension to Indoor Space

In this section, we extend these techniques to process CRSK-mo queries in indoor space. The indoor space can be considered as Euclidean space subdivided into several partitions connected to each other, i.e., indoor partitions such as rooms, hallways and stairs. We show that the techniques which are presented in Section 6.3 can be easily extended to the indoor space by considering the indoor distance rather the Euclidean distance. Thus, we redefine our scoring function as follows.

$$score(q, o) = \alpha.dist(q, o) + (1 - \alpha).S_t(q, o) \tag{6.7}$$

where $dist(q, o)$ is the indoor distance between $q.\lambda$ and $o.\lambda$.

It is easy to see that all the lemmas and pruning rules are correct, even when the indoor distance is used instead of the Euclidean distance.

6.6.1 Safe zone of an object Revisited

In Section 6.3.1, we define the safe zone of an object involving multiple queries as the intersection of the conditional areas of the object with respect to all the CRSK-mo queries. As we mentioned earlier, we must consider the indoor distance instead of the Euclidean distance thus the conditional area of an object with respect to a query cannot be regarded as a circle if it overlaps with neighbourhood partitions. For example, Figure 6.21, shows that an indoor venue consists of four indoor partitions, i.e., P_1, P_2, P_3, P_4 and P_5, and corresponding doors d_1, d_2, d_3, d_4, d_5 and d_6. Let q_1 be a CRSK-mo query and o_1 be an object. Assume that both q_1 and o_1 are located in the partition P_2. The

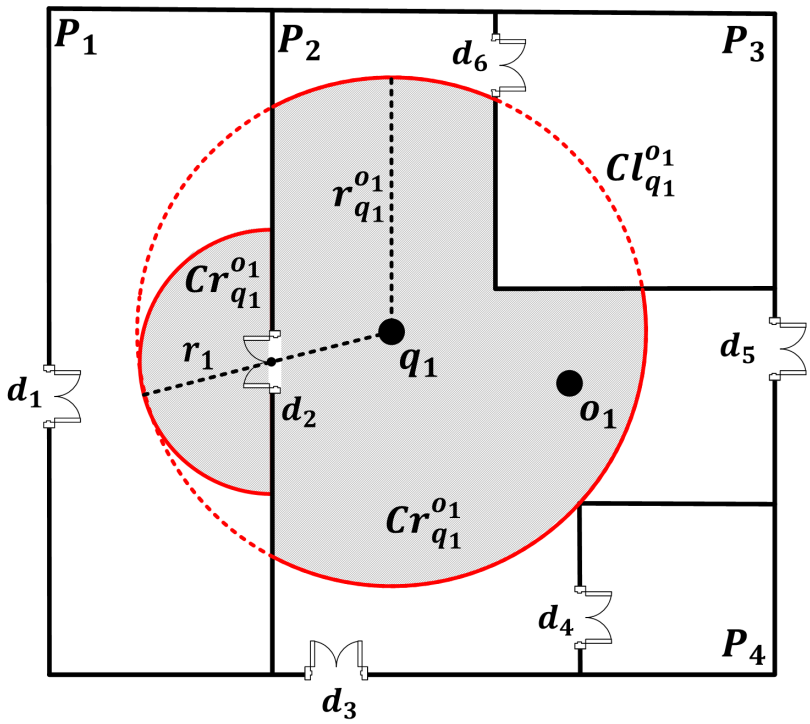


Figure 6.21: Example of a safe zone

corresponding conditional circle of object o_1 with respect to q_1, i.e., $Cl_{q_1}^{o_1}$, is shown in dotted line. Note that the area covered by $Cl_{q_1}^{o_1}$ is no longer a valid safe region as it overlaps with P_1 and P_3 partitions. Hence, the conditional area of object o_1 must be determined considering the indoor distances. Such an area is called a "conditional region" (denoted by $Cr_{q_j}^{o_i}$). The shaded area shown in Figure 6.21 depicts the conditional region of object o_1, i.e., $Cr_{q_1}^{o_1}$. Here, $r_1 = r_{q_1}^{o_1} - dist(q_1, d_2)$ where $dist(q_i, d_i)$ is the indoor distance from query q_i to door d_i. Since the object is within the $Cr_{q_1}^{o_1}$ and one query is available, the $Cr_{q_1}^{o_1}$ is allocated as the safe zone of the object o_1.

Moreover, the intersection of the conditional regions of an object with respect to all the CRSK-mo queries is determined as the safe region of the particular object.

6.6.2 Buffer Region Revisited

A buffer region is formed using $m-1$ conditional regions such that the client device is assigned with a reasonable computational workload. Since the indoor space is partitioned, we can allocate the whole partition as a buffer region only if partition overlaps with $m-1$ conditional regions. Otherwise, we have to determine an area inside the partition that overlaps with at most $m-1$ conditional regions. Figure 6.22 shows three conditional regions of object o_1 with respect to query q_1, q_2 and q_3. The regions boundaries of $Cr_{q_1}^{o_1}$, $Cr_{q_2}^{o_1}$ and $Cr_{q_3}^{o_1}$ are shown in red, green and blue solid lines respectively. As Figure 6.22 illustrates, the whole space of the partition P_2 can be determined as the buffer region of object o_1 only if object capacity is 3. And the area shaded in grey color is assigned as the safe zone of the object o_1. Assume the capacity of the object o_1 is 2. Then the area of the circle centered at o_1 is assigned as the buffer region and the safe zone will be the shaded area within the circle. Moreover, If the capacity of the object is large and the current partition overlaps with less number of conditional regions (i.e., $< m-1$), then the buffer region can be determined by exploring the neighborhood partitions.

6.6.3 Indexing CRSK-mo Queries Revisited

As we already stated, circles may not be valid since the $dist(q, o)$ is determined using the indoor distance. Hence, the area within the r_q^{max} is called "largest conditional region", denoted by Cr_q^{max}. Figure 6.23 shows an example of such a region. The corresponding indoor venue consists of four indoor partitions, i.e., P_1, P_2, P_3, P_4 and P_5, and doors d_1, d_2, d_3, d_4, d_5 and d_6. Let q_1 be a CRSK-mo query. The largest conditional circle $Cl_{q_1}^{max}$, is marked using dotted lines. The area shaded in grey color is the largest conditional region of query q_1 $Cr_{q_1}^{max}$.

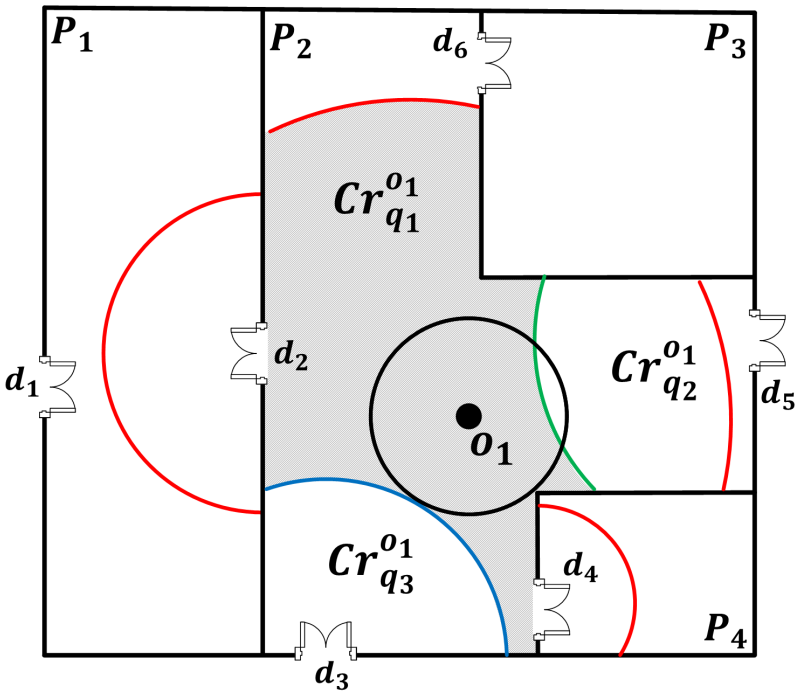


Figure 6.22: Example of a buffer region

Note that $r_1 = r_{q_1}^{max} - dist(q_1, d_2)$ and $r_2 = r_{q_1}^{max} - dist(q_1, d_4)$ accordingly.

We can either use a single grid index or multiple grid indexes (i.e., one grid index per indoor partition) to index the CRSK-mo queries. Hence, the queries can be stored in the particular grid cells as described in Section 6.4.1. Figure 6.23 shows a grid index covering whole indoor space. Hence, the query q_1 is stored at all the grid cells that overlap with $Cr_{q_1}^{max}$ (i.e., partially or fully shaded grid cells).

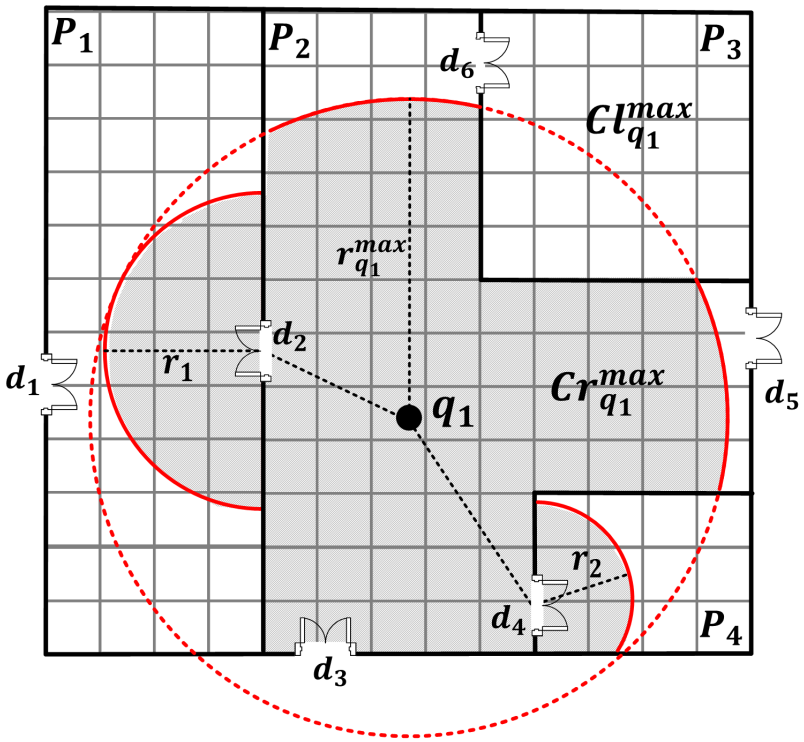


Figure 6.23: Example of marking indoor grid

6.7 Conclusions

In this chapter, we have proposed an efficient solution for processing continuous range spatial keyword queries over moving spatio-textual objects (namely, *CRSK-mo* queries). We exploit the spatial and textual upper bounds between queries and objects to form safe zones and boundary regions to reduce both communication and computation overhead at both ends, i.e., the client-side and the server-side. We have also devised efficient pruning rules to quickly prune objects and queries through smart in-memory data structures for faster query processing. Our experimental results show that our approach achieves high performance and good scalability compared to the competitive PCR approach. Finally, we explain how to extend the proposed techniques to the indoor space.

Chapter 7

Concluding Remarks

7.1 Conclusions

In this thesis, we focus on the urban environment specifically the indoor space and present efficient techniques to process various spatial queries under different settings. We show that using the specialized indexing techniques, it is possible to improve the performance compared to the existing techniques and all our techniques can be applied for indoor as well as outdoor covering the urban areas. Chapter 3 presents our research on route planning queries. We present efficient techniques to answer skyline route planning queries in Chapter 4. Chapter 5 provides our approach to answer continuous detour queries. We discuss our research on continuous spatial keyword queries in Chapter 6. Below are the details.

In Chapter 3 , we study the problem of category aware multi-criteria route planning query, denoted by CAM, which returns a route from a given source indoor point to a target indoor point that passes through at least one indoor point from each given category while minimizing the route cost in terms of

travel and static costs. We show that the problem of answering a CAM query is NP-hard in the number of query categories. We propose two efficient exact solutions, namely BFNE and BFNE-opt to answer CAM queries when the number of query categories is limited. The second solution BFNE-opt is an improvement of BFNE that utilizes a novel traversal method for graph expansion. However, the exact algorithms become very expensive when the number of query categories is large. Hence, we devise an efficient approximation algorithm called GCNN. Later, we propose an improved solution called GCNN-dom which is based on a novel dominance-based pruning technique. It utilizes a pre-processing phase to eliminate all the points that are highly unlikely to be selected in generating an optimal route. The empirical studies on a large real-world dataset demonstrate that the proposed algorithms are highly efficient and offer high-quality results.

In Chapter 4, we study an interesting route planning problem called keyword-aware skyline routes (KSR) query which returns a set of non-dominated routes instead of an optimal route. We consider two attributes in determining the dominance of a route over another route, namely the route distance and the number of partitions that the route visits to cover query keywords. KSR queries facilitate the users to find the most suitable route among the skyline routes based on these dimensions. We prove that the problem of answering a KSR query is NP-hard. We devise an efficient exact algorithm for this problem, assuming that the number of query keywords is small. The results of the empirical studies on a real-world dataset show the efficiency and the scalability of our algorithm.

In Chapter 5, we propose an efficient solution for continuously answering detour queries in the indoor space. We address the problem by solving two individual subproblems, namely, local and remote computation. First, we

introduce a pre-processing approach for efficient local computation that constructs safe zones for indoor objects. Then propose a best first algorithm to efficiently compute a remote detour for a given door of an indoor partition. Finally, we integrate the outcome of the local and remote computations and introduce a client-server framework to answer the continuous detour queries efficiently. The extensive set of experiments show that our proposed approach outperforms the competitive approaches with respect to both computation and communication cost.

In Chapter 6, we have proposed an efficient solution for processing continuous range spatial keyword queries over moving geo-textual objects (namely *CRSK-mo* queries). To efficiently process *CRSK-mo* queries, we have exploited the spatial and textual upper bounds between queries and objects to form safe zones (at the client-side) and boundary regions (at the server-side) to reduce both communication and computation overhead. We have also devised efficient pruning rules to quickly prune objects and queries through smart in-memory data structures for faster processing of queries. Our experimental results show that our approach achieves high performance and good scalability compared to the competitive PCR approach. Finally, we provide a discussion on extending these techniques to the indoor space.

7.2 Directions for Future Work

In this section, we propose several possible directions for future work.

- In Chapter 3, we study category aware multi-criteria route planning queries that take into account the category of each indoor point. However, with the advent of *Web 2.0*, indoor objects are annotated with a set of keywords describing its own substructure. The keywords provide

low level descriptions and they are more specific. Hence, it will be interesting to investigate route planning queries that take into account indoor objects consist of multiple keywords in route planning where the textual similarity is determined as another criterion to compute the route cost in addition to the current criteria.

- In Chapter 5, we study continuous detour queries in indoor venues. Another attractive indoor LBS is the continuous nearest neighbor queries. For example, a user may be interested in finding the nearby ATM while she is doing some window shopping. The proposed pre-processing techniques that facilitate efficient local computations in query time can be extended to answer continuous nearest neighbor queries.

- We focus mainly on the case where the indoor space does not have any updates. In real-world, some properties of indoor space may change over time. For example, some doors are closed after hours or indoor objects are moved. So it is an important direction to see how to update the existing indexes such as VIP-tree efficiently or alternative indexes which are more efficient to update.

Bibliography

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

[2] Jin Soung Yoo and Shashi Shekhar. In-route nearest neighbor queries. *GeoInformatica*, 9(2):117–137, 2005.

[3] Sarana Nutanong, Egemen Tanin, Jie Shao, Rui Zhang, and Ramamohanarao Kotagiri. Continuous detour queries in spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, 24(7):1201–1215, 2012.

[4] Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang, and Wenjie Zhang. Lazy updates: An efficient technique to continuously monitoring reverse knn. *Proceedings of the VLDB Endowment*, 2(1):1138–1149, 2009.

[5] Haosheng Huang and Georg Gartner. Current trends and challenges in location-based services, 2018.

[6] `https://www.acma.gov.au/theACMA/Library/researchacma/Research-reports/here-there-and-everywhere-consumer-behaviour-and-location-services`.

[7] https://www.gsa.europa.eu/library/case-studies/location-based-services-lbs-mobile-applications.

[8] https://www.computerworld.com.au/article/423522/location-based_services_they_there_yet_/.

[9] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 802–813. VLDB Endowment, 2003.

[10] Mohammad R Kolahdouzan and Cyrus Shahabi. Alternative solutions for continuous k nearest neighbor queries in spatial network databases. *GeoInformatica*, 9(4):321–341, 2005.

[11] Mohammad Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 840–851. VLDB Endowment, 2004.

[12] Hyung-Ju Cho and Chin-Wan Chung. An efficient and scalable approach to cnn queries in a road network. In *Proceedings of the 31st international conference on Very large data bases*, pages 865–876. VLDB Endowment, 2005.

[13] Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):820–833, 2005.

[14] Pengfei Zhang, Huaizhong Lin, Yunjun Gao, and Dongming Lu. Aggregate keyword nearest neighbor queries on road networks. *GeoInformatica*, 22(2):237–268, 2018.

[15] Buğra Gedik and Ling Liu. Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *Advances in Database Technology-EDBT 2004*, pages 67–87. Springer, 2004.

[16] Haojun Wang, Roger Zimmermann, and Wei-Shinn Ku. Distributed continuous range query processing on moving objects. In *International Conference on Database and Expert Systems Applications*, pages 655–665. Springer, 2006.

[17] Ying Cai, Kien A Hua, and Guohong Cao. Processing range-monitoring queries on heterogeneous mobile objects. In *IEEE International Conference on Mobile Data Management, 2004. Proceedings. 2004*, pages 27–38. IEEE, 2004.

[18] Tongyu Zhu, Chen Wang, Weifeng Lv, and Jian Huang. Continuous range monitoring of moving objects in road networks. In *2010 10th International Conference on Intelligent Systems Design and Applications*, pages 1412–1417. IEEE, 2010.

[19] Wei Wu, Fei Yang, Chee-Yong Chan, and Kian-Lee Tan. Finch: Evaluating reverse k-nearest-neighbor queries on location data. *Proceedings of the VLDB Endowment*, 1(1):1056–1067, 2008.

[20] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 577–588. IEEE, 2011.

[21] Muhammad Aamir Cheema, Xuemin Lin, Wei Wang, Wenjie Zhang, and Jian Pei. Probabilistic reverse nearest neighbor queries on uncer-

tain data. *IEEE Transactions on Knowledge and Data Engineering*, 22(4):550–564, 2010.

[22] Elke Achtert, Hans-Peter Kriegel, Peer Kröger, Matthias Renz, and Andreas Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 886–897. ACM, 2009.

[23] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. Hybrid index structures for location-based web search. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 155–162. ACM, 2005.

[24] Subodh Vaid, Christopher B Jones, Hideo Joho, and Mark Sanderson. Spatio-textual indexing for geographical search on the web. In *International Symposium on Spatial and Temporal Databases*, pages 218–235. Springer, 2005.

[25] Dongxiang Zhang, Kian-Lee Tan, and Anthony KH Tung. Scalable top-k spatial keyword search. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 359–370. ACM, 2013.

[26] Dingming Wu, Man Lung Yiu, Christian S Jensen, and Gao Cong. Efficient continuously moving top-k spatial keyword query processing. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 541–552. IEEE, 2011.

[27] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. Ap-tree: Efficiently support continuous spatial-keyword queries over

stream. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1107–1118. IEEE, 2015.

[28] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 421–430. IEEE, 2001.

[29] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 275–286. VLDB Endowment, 2002.

[30] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 467–478. ACM, 2003.

[31] Ke Deng, Xiaofang Zhou, and Heng Tao. Multi-source skyline query processing in road networks. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 796–805. IEEE, 2007.

[32] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. Route skyline queries: A multi-preference path planning approach. 2010.

[33] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In *SSTD*, volume 5, pages 273–290. Springer, 2005.

[34] Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *The VLDB Journal—The International Journal on Very Large Data Bases*, 17(4):765–787, 2008.

[35] Xin Cao, Lisi Chen, Gao Cong, and Xiaokui Xiao. Keyword-aware optimal route search. *Proceedings of the VLDB Endowment*, 5(11):1136–1147, 2012.

[36] Yifeng Zeng, Xuefeng Chen, Xin Cao, Shengchao Qin, Marc Cavazza, and Yanping Xiang. Optimal route search with the coverage of users' preferences. In *IJCAI*, pages 2118–2124, 2015.

[37] Bin Yao, Mingwang Tang, and Feifei Li. Multi-approximate-keyword routing in gis data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 201–210. ACM, 2011.

[38] Shashi Shekhar and Jin Soung Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 9–16. ACM, 2003.

[39] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, and Jeffrey Xu Yu. Monitoring path nearest neighbor in road networks. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 591–602. ACM, 2009.

[40] Shuo Shang, Ke Deng, and Kexin Xie. Best point detour query in road networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 71–80. ACM, 2010.

[41] Peggy L Jenkins, Thomas J Phillips, Elliot J Mulberg, and Steve P Hui. Activity patterns of californians: use of and proximity to indoor

pollutant sources. *Atmospheric Environment. Part A. General Topics*, 26(12):2141–2148, 1992.

[42] Neil E Klepeis, William C Nelson, Wayne R Ott, John P Robinson, Andy M Tsang, Paul Switzer, Joseph V Behar, Stephen C Hern, and William H Engelmann. The national human activity pattern survey (nhaps): a resource for assessing exposure to environmental pollutants. *Journal of Exposure Science and Environmental Epidemiology*, 11(3):231, 2001.

[43] Lisi Chen, Gao Cong, Christian S Jensen, and Dingming Wu. Spatial keyword query processing: an experimental evaluation. In *Proceedings of the VLDB Endowment*, volume 6, pages 217–228. VLDB Endowment, 2013.

[44] Bin Yang, Hua Lu, and Christian S Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *Proceedings of the 13th international conference on extending database technology*, pages 335–346. ACM, 2010.

[45] Rudra Ranajee Saha, Tanzima Hashem, Tasmia Shahriar, and Lars Kulik. Continuous obstructed detour queries. In *10th International Conference on Geographic Information Science (GIScience 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[46] Hua Lu, Bin Yang, and Christian S Jensen. Spatio-temporal joins on symbolic indoor tracking data. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 816–827. IEEE, 2011.

[47] Xike Xie, Hua Lu, and Torben Bach Pedersen. Distance-aware join for indoor moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):428–442, 2015.

[48] Bin Yang, Hua Lu, and Christian S Jensen. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 671–680. ACM, 2009.

[49] Wenjie Yuan and Markus Schneider. Supporting continuous range queries in indoor space. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on*, pages 209–214. IEEE, 2010.

[50] Christian S Jensen, Hua Lu, and Bin Yang. Indexing the trajectories of moving objects in symbolic indoor space. In *SSTD*, pages 208–227. Springer, 2009.

[51] Xike Xie, Hua Lu, and Torben Bach Pedersen. Efficient distance-aware query evaluation on indoor moving objects. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 434–445. IEEE, 2013.

[52] Hua Lu, Xin Cao, and Christian S Jensen. A foundation for efficient indoor distance-aware query processing. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 438–449. IEEE, 2012.

[53] Tenindra Abeywickrama, Muhammad Aamir Cheema, and David Taniar. K-nearest neighbors on road networks: a journey in experimentation and in-memory implementation. *Proceedings of the VLDB Endowment*, 9(6):492–503, 2016.

[54] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, Lizhu Zhou, and Zhiguo Gong. G-tree: An efficient and scalable index for spatial search on road networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(8):2175–2189, 2015.

[55] Ken CK Lee, Wang-Chien Lee, Baihua Zheng, and Yuan Tian. Road: A new spatial object search framework for road networks. *IEEE transactions on knowledge and data engineering*, 24(3):547–560, 2012.

[56] Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. Efficient query processing on spatial networks. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 200–209. ACM, 2005.

[57] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. G-tree: An efficient index for knn search on road networks. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 39–48. ACM, 2013.

[58] Tenindra Abeywickrama and Muhammad Aamir Cheema. Efficient landmark-based candidate generation for knn queries on road networks. In *International Conference on Database Systems for Advanced Applications*, pages 425–440. Springer, 2017.

[59] Yaron Kanza, Eliyahu Safra, and Yehoshua Sagiv. Route search over probabilistic geospatial data. In *SSTD*, volume 9, pages 153–170. Springer, 2009.

[60] Yaron Kanza, Roy Levin, Eliyahu Safra, and Yehoshua Sagiv. Interactive route search in the presence of order constraints. *Proceedings of the VLDB Endowment*, 3(1-2):117–128, 2010.

[61] Mehdi Sharifzadeh and Cyrus Shahabi. Processing optimal sequenced route queries using voronoi diagrams. *GeoInformatica*, 12(4):411–433, 2008.

[62] Haiquan Chen, Wei-Shinn Ku, Min-Te Sun, and Roger Zimmermann. The partial sequenced route query with traveling rules in road networks. *GeoInformatica*, 15(3):541–569, 2011.

[63] Jing Li, Yin David Yang, and Nikos Mamoulis. Optimal route queries with arbitrary order constraints. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1097–1110, 2013.

[64] Haiquan Chen, Wei-Shinn Ku, Min-Te Sun, and Roger Zimmermann. The multi-rule partial sequenced route query. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 10. ACM, 2008.

[65] Zhou Shao, Muhammad Aamir Cheema, and David Taniar. Trip planning queries in indoor venues. *The Computer Journal*, 61:1–18, 2017.

[66] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting: Theory and optimizations. In *Intelligent Information Processing and Web Mining*, pages 595–604. Springer, 2005.

[67] Kian-Lee Tan, Pin-Kwang Eng, Beng Chin Ooi, et al. Efficient progressive skyline computation. In *VLDB*, volume 1, pages 301–310, 2001.

[68] Wenjie Zhang, Aiping Li, Muhammad Aamir Cheema, Ying Zhang, and Lijun Chang. Probabilistic n-of-n skyline computation over uncertain data streams. *World Wide Web*, 18(5):1331–1350, 2015.

[69] Wenjie Zhang, Muhammad Aamir Cheema, Ying Zhang, and Xuemin Lin. Skyline: Stacking optimal solutions in exact and uncertain worlds. *Int. J. Software and Informatics*, 6(4):475–493, 2012.

[70] Xuemin Lin, Yidong Yuan, Wei Wang, and Hongjun Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *21st International Conference on Data Engineering (ICDE'05)*, pages 502–513. IEEE, 2005.

[71] Yuan Tian, Ken CK Lee, and Wang-Chien Lee. Finding skyline paths in road networks. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 444–447. ACM, 2009.

[72] Xuegang Huang and Christian S Jensen. In-route skyline querying for location-based services. In *International Workshop on Web and Wireless Geographical Information Systems*, pages 120–135. Springer, 2004.

[73] Sumin Jang and Jaesoo Yoo. Processing continuous skyline queries in road networks. In *Computer Science and its Applications, 2008. CSA'08. International Symposium on*, pages 353–356. IEEE, 2008.

[74] Yuan-Ko Huang, Chia-Heng Chang, and Chiang Lee. Continuous distance-based skyline queries in road networks. *Information Systems*, 37(7):611–633, 2012.

[75] Wan Ting Hsu, Yu Ting Wen, Ling Yin Wei, and Wen Chih Peng. Skyline travel routes: Exploring skyline for trip planning. In *2014 IEEE 15th International Conference on Mobile Data Management*, volume 2, pages 31–36. IEEE, 2014.

[76] Saad Aljubayrin, Zhen He, and Rui Zhang. Skyline trips of multiple pois categories. In *International Conference on Database Systems for Advanced Applications*, pages 189–206. Springer, 2015.

[77] Simonas Šaltenis. Indexing the positions of continuously moving objects. In *Encyclopedia of GIS*, pages 538–543. Springer, 2008.

[78] Yufei Tao, Dimitris Papadias, and Jimeng Sun. The tpr*-tree: an optimized spatio-temporal access method for predictive queries. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 790–801. VLDB Endowment, 2003.

[79] Sunil Prabhakar, Yuni Xia, Dmitri V Kalashnikov, Walid G Aref, and Susanne E Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *Computers, IEEE Transactions on*, 51(10):1124–1140, 2002.

[80] K-L Wu, S-K Chen, and Philip S Yu. On incremental processing of continual range queries for location-aware services and applications. In *The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 261–269. IEEE, 2005.

[81] MoonBae Song, Hyunseung Choo, and Won Kim. Spatial indexing for massively update intensive applications. *Information Sciences*, 203:1–23, 2012.

[82] Haibo Hu, Jianliang Xu, and Dik Lun Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 479–490. ACM, 2005.

[83] Ying Cai, Kien A Hua, Guohong Cao, and Toby Xu. Real-time processing of range-monitoring queries in heterogeneous mobile databases. *IEEE Transactions on Mobile Computing*, 5(7):931–942, 2006.

[84] HaRim Jung, Yong Sung Kim, and Yon Dohn Chung. Qr-tree: An efficient and scalable method for evaluation of continuous range queries. *Information Sciences*, 274:156–176, 2014.

[85] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. Continuous monitoring of distance-based range queries. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1182–1199, 2011.

[86] Xiaohui Yu, Ken Q Pu, and Nick Koudas. Monitoring k-nearest neighbor queries over moving objects. In *21st International Conference on Data Engineering (ICDE'05)*, pages 631–642. IEEE, 2005.

[87] Kyriakos Mouratidis, Dimitris Papadias, and Marios Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 634–645. ACM, 2005.

[88] Kyriakos Mouratidis, Dimitris Papadias, Spiridon Bakiras, and Yufei Tao. A threshold-based algorithm for continuous monitoring of k nearest neighbors. *Knowledge and Data Engineering, IEEE Transactions on*, 17(11):1451–1464, 2005.

[89] Muhammad Aamir Cheema, Yidong Yuan, and Xuemin Lin. Circular-trip: an effective algorithm for continuous knn queries. In *International*

Conference on Database Systems for Advanced Applications, pages 863–869. Springer, 2007.

[90] Sarana Nutanong, Rui Zhang, Egemen Tanin, and Lars Kulik. Analysis and evaluation of v*-knn: an efficient algorithm for moving knn queries. *The VLDB Journal*, 19(3):307–332, 2010.

[91] Wei-Wei Sun, Chu-Nan Chen, Liang Zhu, Yun-Jun Gao, Yi-Nan Jing, and Qing Li. On efficient aggregate nearest neighbor query processing in road networks. *Journal of computer science and technology*, 30(4):781–798, 2015.

[92] Liang Zhu, Yinan Jing, Weiwei Sun, Dingding Mao, and Peng Liu. Voronoi-based aggregate nearest neighbor query processing in road networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 518–521. ACM, 2010.

[93] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM computing surveys (CSUR)*, 38(2):6, 2006.

[94] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. Citeseer, 1990.

[95] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. Keyword search on spatial databases. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 656–665. IEEE, 2008.

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

[97] Zhisheng Li, Ken CK Lee, Baihua Zheng, Wang-Chien Lee, Dik Lee, and Xufa Wang. IR-tree: An efficient index for geographic document search. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):585–599, 2011.

[98] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S Jensen. Joint top-k spatial keyword query processing. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1889–1903, 2012.

[99] João B Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Advances in Spatial and Temporal Databases*, pages 205–222. Springer, 2011.

[100] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. Efficient query processing in geographic web search engines. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 277–288. ACM, 2006.

[101] Maria Christoforaki, Jinru He, Constantinos Dimopoulos, Alexander Markowetz, and Torsten Suel. Text vs. space: efficient geo-search query processing. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 423–432. ACM, 2011.

[102] Ali Khodaei, Cyrus Shahabi, and Chen Li. Hybrid indexing and seamless ranking of spatial and textual features of web documents. In *Interna-*

tional Conference on Database and Expert Systems Applications, pages 450–466. Springer, 2010.

[103] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Inverted linear quadtree: Efficient top k spatial keyword search. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1706–1721, 2016.

[104] Weihuang Huang, Guoliang Li, Kian-Lee Tan, and Jianhua Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 932–941. ACM, 2012.

[105] Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony KH Tung, and Masaru Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *2009 IEEE 25th International Conference on Data Engineering*, pages 688–699. IEEE, 2009.

[106] Long Guo, Jie Shao, Htoo Htet Aung, and Kian-Lee Tan. Efficient continuous top-k spatial keyword queries on road networks. *GeoInformatica*, 19(1):29–60, 2015.

[107] Xin Cao, Gao Cong, Christian S Jensen, and Beng Chin Ooi. Collective spatial keyword querying. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 373–384. ACM, 2011.

[108] Long Guo, Dongxiang Zhang, Guoliang Li, Kian-Lee Tan, and Zhifeng Bao. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *Proceedings of the 2015 ACM SIGMOD*

International Conference on Management of Data, pages 843–857. ACM, 2015.

[109] Lisi Chen, Gao Cong, Xin Cao, and Kian-Lee Tan. Temporal spatial-keyword top-k publish/subscribe. In *2015 IEEE 31st International Conference on Data Engineering*, pages 255–266. IEEE, 2015.

[110] Huiqi Hu, Yiqun Liu, Guoliang Li, Jianhua Feng, and Kian-Lee Tan. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *2015 IEEE 31st International Conference on Data Engineering*, pages 711–722. IEEE, 2015.

[111] Chaluka Salgado, Muhammad Aamir Cheema, and David Taniar. An efficient approximation algorithm for multi-criteria indoor route planning queries. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 448–451. ACM, 2018.

[112] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 421–430. ACM, 2012.

[113] Davide Dardari, Pau Closas, and Petar M Djuric. Indoor tracking: Theory, methods, and technologies. *IEEE Trans. Vehicular Technology*, 64(4):1263–1278, 2015.

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

[115] Kyriakos Mouratidis, Yimin Lin, and Man Lung Yiu. Preference queries in large multi-cost transportation networks. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 533–544. IEEE, 2010.

[116] Chaluka Salgado, Muhammad Aamir Cheema, and Mohammed Eunus Ali. Continuous monitoring of range spatial keyword query over moving objects. *World Wide Web*, 21(3):687–712, 2018.

[117] Muhammad Aamir Cheema, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Xuefei Li. Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. *The VLDB Journal—The International Journal on Very Large Data Bases*, 21(1):69–95, 2012.

[118] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. A safe zone based approach for monitoring moving skyline queries. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 275–286. ACM, 2013.

[119] Mohamed F Mokbel, Xiaopeing Xiong, and Walid G Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 623–634. ACM, 2004.

[120] Thomas Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.