SPATIAL KEYWORD QUERY PROCESSING OVER ROAD NETWORKS

WENGEN LI

Ph.D

The Hong Kong Polytechnic University

This programme is jointly offered by The Hong Kong Polytechnic University and Tongji University

2018

The Hong Kong Polytechnic University
Department of Computing

Tongji University
Department of Computer Science and Technology

Spatial Keyword Query Processing over Road Networks

Wengen LI

A thesis submitted in partial fulfilment of the requirements

for the degree of Doctor of Philosophy

May 2017

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Wengen LI (Name of student)

Abstract

With the rapid development of geo-positioning techniques and high-speed mobile networks, location-aware applications like location-based services, location-based social networks and intelligent driving navigation have been gaining tremendous popularity in recent years. When using these location-aware applications, users often issue queries with both spatial and textual requirements, e.g., searching restaurants within one kilometer from the current location, where the term "restaurants" is textual requirement while "within one kilometer from the current location" is spatial requirement. To provide users query services regarding both spatial and textual requirements, the research community has proposed ***Spatial Keyword Query*** which combines traditional spatial query in spatial databases and keyword query in information retrieval.

Specifically, given a spatial keyword query with spatial requirement (e.g., query location and spatial range) and textual requirement (usually described by query keywords), both of (i) the spatial relationship (e.g., spatial proximity) between target objects and the spatial requirement and (ii) the textual relevance between the textual descriptions of target objects and query keywords are considered. In the past decade, extensive studies have been conducted to solve all kinds of spatial keyword queries. However, these studies mainly focus on Euclidean space and cannot effectively support spatial keyword queries over road networks on which the spatial proximity between two locations is network distance rather than Euclidean distance. Considering that people's travel routes are restricted by road networks, and the Euclidean distance and network distance between two locations could be quite different, it is of high necessity to study spatial keyword query over road networks. In addition, existing spatial keyword queries mainly aim to locate the target objects and ignore the function of routing, and are thus unable to provide complete travel solutions

to get to the target objects. Motivated by these observations, this thesis focuses on"***Spatial Keyword Query Processing over Road Networks***" and aims to propose effective solutions to the following three typical types of interesting queries:

First, we study services locating over road networks to find the locations of required services. For this type of query, we propose ***Range Spatial Keyword (RSK) query*** to retrieve all the objects that are textually relevant to the query keywords and locate within a given range from the query location. Though RSK query has received extensive studies in Euclidean space, little has been done to deal with it over road networks. To process RSK query over road networks efficiently, we first propose an expansion-based approach based on the locality of RSK query. Then, we improve the efficiency of this approach based on the observation that road network distance is always larger than or equal to the corresponding Euclidean distance between two locations. In addition, to ensure high scalability on large road networks and RSK queries of large query ranges, we design an Rnet Hierarchy index and devise an efficient query processing algorithm based on this index. According to the experiments, Rnet Hierarchy-based approach can deal with RSK queries over road networks of millions of vertices.

Second, to search a route relevant to query keywords, we propose two queries, i.e., ***Keyword Coverage Route (KCR) query*** and ***Bounded-Cost Information Route (BCIR) query***. Given a set of query keywords, KCR query retrieves an optimal route such that its length is less than a distance threshold and it covers the most query keywords. By contrast, BCIR query retrieves the optimal route whose textual relevance to the query keywords is maximized and cost (e.g., length and travel time) is less than a cost budget. Different from KCR query, BCIR query considers general cost rather than the spatial distance and aims to maximize the global textual relevance rather than maximizing the number of query keywords on the routes. KCR query is particularly helpful for tourists to search the routes covering many interested objects while BCIR query is useful for tourists and city explorers to search for routes of specific topics. Both KCR query and BCIR query are NP-hard problems without solutions of polynominal time complexity. To solve KCR query, we

propose an adaptive route sampling framework under which both static and dynamic route sampling techniques are proposed. Particularly, the dynamic route sampling can obtain routes of high quality by learning knowledge from the routes sampled previously. The proposed framework can flexibly compute query results of different qualities according to the response time limit, thus avoiding the low efficiency of traditional exact solutions and the low quality of approximate solutions. To solve BCIR query, we propose different solutions for different application scenarios. On the one hand, we design an exact solution for the BCIR queries of small cost budgets and propose multiple pruning techniques to reduce the searching space. On the other hand, we propose a time-bounded solution and an error-bounded solution for BCIR queries of large cost budgets. Time-bounded solution initializes a set of candidate routes and keeps refining them until reaching the given response time limit. Error-bounded solution is adapted from the exact solution by relaxing the pruning requirement to improve the pruning efficiency and can greatly reduce the query processing time while guaranteeing the quality of query results.

Third, considering that the travel times of routes are uncertain and dynamically changing over time, we propose uncertain road network model which represents the travel time of each road by a dynamic discrete random variable. Then, we propose the ***Probabilistic Time-constrained Route (PTR) query*** to retrieve keyword-aware routes over this model such that (i) the returned routes sequentially pass multiple categories of POIs (points of interest, e.g., bank and restaurant) according to the order of given query keywords and (ii) the travel times of routes are small in high confidence. To answer PTR queries efficiently, we propose a two-phase query approach which first generates a small set of candidate routes by employing effective pruning strategies regarding the service time constraints on POIs and the probabilistic/rank requirements of queries. A refinement operation based on Monte Carlo sampling is then conducted over the candidate routes to compute the final results.

List of Publications

1. **Wengen Li**, Jiannong Cao, Jihong Guan, Man Lung Yiu and Shuigeng Zhou, "Efficient Retrieval of Bounded-Cost Informative Routes", IEEE Transactions on Knowledge and Data Engineering (TKDE), 29(10): 2182-2196, 2017

2. **Wengen Li**, Jihong Guan, Xiang Lian, Shuigeng Zhou and Jiannong Cao, "Probabilistic Time-Constrained Paths Search over Uncertain Road Networks", IEEE Transactions on Services Computing (TSC), DOI: 10.1109/TSC.2016.2582692, 2016

3. **Wengen Li**, Jiannong Cao, Jihong Guan, Man Lung Yiu and Shuigeng Zhou, "Retrieving Routes of Interest Over Road Networks", in WAIM, 2016, pp. 109-123

4. **Wengen Li**, Jihong Guan and Shuigeng Zhou, "Efficiently Evaluating Range-Constrained Spatial Keyword Query on Road Networks", in DASFAA-SIM3, 2014, pp. 283-295

5. Yuqi Wang, Jiannong Cao, Lifang He, **Wengen Li**, Lichao Sun, Philip S. Yu, "Coupled Sparse Matrix Factorization for Response Time Prediction in Logistics Services", CIKM, 939-947, 2017

6. Yuqi Wang, Jiannong Cao, **Wengen Li**, Tao Gu and Wenzhong Shi, "Exploring traffic congestion correlation from multiple data sources", Pervasive and Mobile Computing Journal (PMCJ),41: 470-483, 2017

7. Yuqi Wang, Jiannong Cao, **Wengen Li** and Tao Gu, "Mining Traffic Congestion Correlation between Road Segments on GPS Trajectories", in SMARTCOMP, 2016, pp. 1-8

Acknowledgements

First of all, I would like to show my great gratitude to Prof. Jiannong Cao, my supervisor at PolyU. His continuous support and encouragement contribute a lot to this thesis. Prof. Cao always has good insights into the hot research topics and gives me so many important suggestions and inspirations. Prof. Cao often encourages us to conduct impactful research, which not only reshapes my attitude to the research work but also stimulates me to improve myself constantly.

I also would like to thank my co-supervisor Prof. Jihong Guan at Tongji University. She provides me so many opportunities to visit other universities and attend international conferences, which greatly expands my research horizon. More importantly, she gives me a lot of help in finding the research topic and leads me to the world of scientific research. In addition, I really appreciate the guide and help from Prof. Shuigeng Zhou who helps me to revise my first paper and gives me many thoughtful comments on my research. I also would like to thank Dr. Man Lung Yiu, Dr. Xiang Lian, and Dr. Gao Cong for their valuable comments and suggestions on my research work.

I really want to take this chance to thank my parents who are always there supporting and encouraging me. I can seldom go back home during my PhD study. Thanks for your understanding. I also would like to thank my grandparents and my sister. Thanks for your love and giving me the warmth of home.

In addition, I would like to thank my dear research colleagues like Dr. Yichao Zhang, Mr. Wei Xu, Mr. Weidan Li, Mr. Haiquan Zhu, Ms. Xiaomin Wei, Mr. Yuqi Wang, Mr. Rui Liu, Dr. Lei Yang, Prof. Junbin Liang, Dr. Quincy Liang, Mr. Jiaxing Shen, Dr. Xiulong Liu, Dr. Fuliang Li and many more. I learn so much from the innumerable meetings and talks with them.

My dear friends, Mr. Yumeng Guo, Mr. Qiang Li, Mr. Ruosong Yang, Dr.

Wei Lu, Mr. Lihui Zheng, Mr. Rongguo Li, Mr. Zhijia Zhang and many others, I really appreciate all kinds of help and support from you. Thanks for enriching and brightening my life.

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

In recent years, location-aware applications receive tremendous popularity due to the wide spread of smart mobile devices and the rapid development of techniques in geo-positioning and mobile communication. For example, navigation systems are widely used for planning routes by drivers; location-based services (LBSs) provide services ranging from entertainment to accommodation; location-based social networks bridge the virtual social networks and the physical world; and geo-marketing brings more convenient and efficient business promotion. Users often issue queries with both spatial and textual attributes when using these location-aware applications. For example, a query searching all the restaurants within one kilometer from the current location has spatial attribute "*within one kilometer from the current location*" and textual attribute "*restaurants*". Another example is to search the k nearest banks where "*k nearest*" and "*banks*" are spatial and textual attributes, respectively. Therefore, we need an efficient way to process queries with both spatial and textual attributes, which is one common concern for many location-aware applications.

The research community has identified this issue and proposed spatial keyword query [18] which combines traditional spatial query in spatial database and keyword

query in information retrieval. Specifically, a spatial keyword query considers both of the spatial proximity between target query objects and query location and the textual relevance between the textual descriptions of target objects and query keywords. Here, the objects could be any spatio-textual data that contains spatial and textual attributes, e.g., Points of Interest (POIs) with locations and textual service descriptions from Google map, geo-tagged comments from Foursquare, geo-tagged photos with text tags from Facebook, geo-tagged posts from twitter, and location-based advertisements from different shopping Web sites. For example, Figure 1.1 illustrates the distribution of POIs in New York and those POIs with textual descriptions on the island of Richmond County in New York, where lines in the background represent road network and points represent all kinds of physical objects like hotels, restaurants, banks, and cinemas.

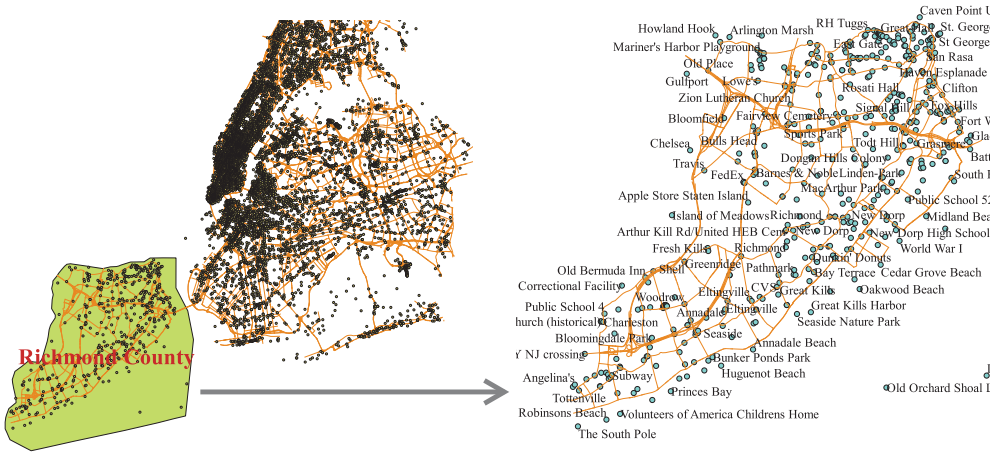


Figure 1.1: The distribution of POIs in New York and the POIs with textual descriptions in Richmond County.

In the past decade, a considerable number of spatial keyword queries based on Euclidean distance have been proposed. In these queries, spatio-textual data is represented as individual spatio-textual objects and each spatio-textual object contains a spatial location and a piece of textual description consisting of keywords. Classical spatial keyword queries include range spatial keyword query and top-k spatial key-

word query, where range spatial keyword query retrieves those spatio-textual objects that are located within some specified query region and contain the given query keywords, top-k spatial keyword query searches for the k optimal spatio-textual objects regarding the spatial proximity and the textual relevance between spatio-textual objects and queries. In addition, some specialized spatial keyword queries such as group spatial keyword query, moving spatial keyword query, and reverse kNN spatial keyword query are also proposed. The major challenge for processing spatial keyword query is to deal with spatial query and keyword query simultaneously. To solve this issue, different indexing methods and query processing algorithms are proposed.

Most existing studies on spatial keyword query are conducted in Euclidean space where the spatial proximity between two locations is quantified by the Euclidean distance. Spatial keyword queries in Euclidean space are useful in Web search engines when spatial factor is involved. In reality, however, people's travel routes are usually strictly constrained by the topology structure of road network, especially for car drivers. In many cases, the Euclidean distance and the network distance between two locations could be quite different. Therefore, it is of high necessity to study spatial keyword query over road networks on which the spatial proximity between two locations is measured by the network distance rather than the Euclidean distance.

Although there are already some studies on spatial keyword query over road networks, it is still far from enough to cover all the existing types of spatial keyword queries and many query problems remain unsolved. Compared with computing the Euclidean distance between two locations, computing network distance is much more time-consuming, thus increasing the difficulty to efficiently process the counterpart spatial keyword queries over road networks. In addition, when it comes to road networks, many new types of spatial keyword queries associated with route search are emerging. For example, a user may issue a query to find a route that covers a set of query keywords. These new queries should also be studied specifically.

With the observations above, we thus propose the topic "***Spatial Keyword Query Processing over Road Networks***" and study three typical types of queries as discussed in Section 1.2.

1.2 Research Scope

This research topic will mainly focus on the following three types of spatial keyword queries over road networks.

First, to effectively locate services, we propose *Range Spatial Keyword* (RSK) query over road networks which searches for all the spatio-textual objects with textual descriptions relevant to the query keywords and locations within a specified area. Though RSK query has received extensive studies in Euclidean space, little has been done to deal with it over road networks. To process RSK query over road networks efficiently, we propose alternative approaches with different indexing strategies and query processing algorithms.

Second, to search a route relevant to query keywords, propose *Keyword Coverage Route* (KCR) query and *Bounded-Cost Informative Route* (BCIR) query. KCR query allows users to specify their interests by query keywords and returns a route that has a distance less than a distance threshold and covers the most number of query keywords. KCR query is particularly helpful for tourists. For example, a tourist may wish to find a route from a scenic spot to her hotel to cover many local artware shops. To make full use of spatio-textual data and provide users an easy way to describe their preferences for travel routes, BCIR query retrieves the routes that are most textually relevant to user-specified query keywords while satisfying a travel cost budget. BCIR query treats the textual description of each route as a document and leverages an information retrieval model to compute its textual relevance to the query keywords, thus different from KCR query.

Third, in practice, traffic conditions over road networks are inherently uncertain and dynamically varying over time, which makes it rather challenging to provide accurate results for route queries based on travel time. Inspired by this observation, we consider the practical settings of road networks and model them by *uncertain road networks (URNs)* on which the travel time of each road is uncertain and captured by a set of travel time samples. Then, we formalize the *Probabilistic Time-constrained Route* (PTR) query over uncertain road networks to retrieve those routes that not only cover required spatio-textual objects with constrained service time but also have the minimum travel times in high confidence.

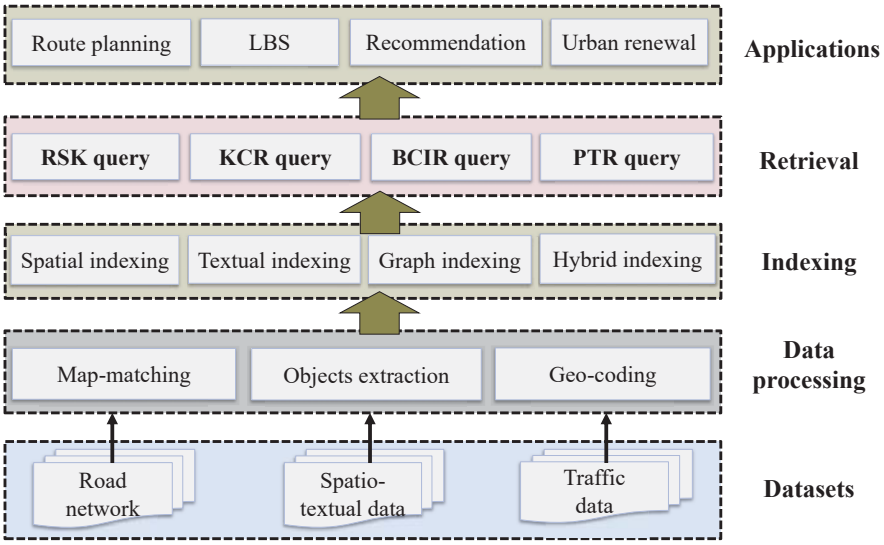


Figure 1.2: Research framework for spatial keyword query processing over road networks.

Figure 1.2 presents the research framework for this topic. For each spatial keyword query, we need to design a suite of specific techniques for data processing, indexing and query processing altogether. Then, the queries can be applied in different application scenarios. It is worth noting that many spatial keyword queries over road networks are NP-hard. For example, all KCR query, BCIR query and PTR query are NP-hard problems, which will be proved in the corresponding chapters. As

of writing this thesis, we cannot expect an exact processing algorithm of polynomial time complexity for these three queries since P=NP remains unproved. Therefore, to efficiently process these queries, we aim to design either exact query processing algorithms with effective pruning techniques or approximate query processing algorithms with high efficiency and quality guarantee.

1.3 Organization

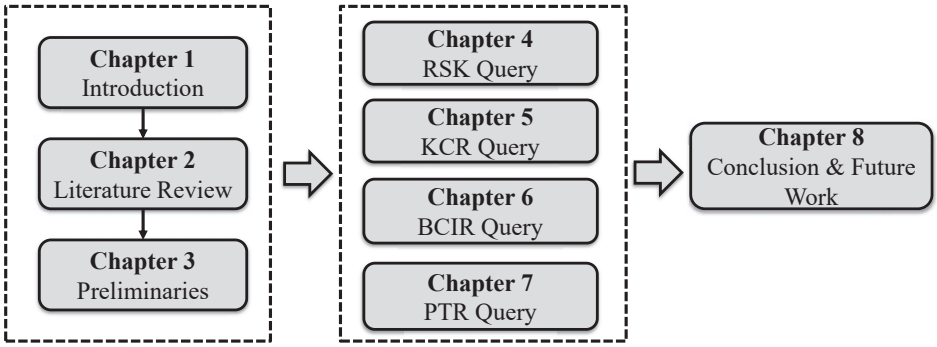


Figure 1.3: The organization of this thesis.

As illustrated in Figure 1.3, this thesis contains eight chapters where

- **Chapter 1** introduces the background, highlights the motivations for studying spatial keyword query over road networks, and overviews the research scope;
- **Chapter 2** reviews the related work and discusses the gap between the practical requirement for spatial keyword query processing and the existing techniques;
- **Chapter 3** provides the definitions of road network and spatio-textual objects, briefly introduces the basic indexing structures that will be used for facilitating query processing, and summarizes the datasets used for evaluating the performance of proposed solutions;

- **Chapter 4** proposes range spatial keyword (RSK) query and designs three approaches with different indexing structures based on the properties of road network and RSK query;

- **Chapter 5** proposes keyword coverage route (KCR) query, analyzes its hardness, and proposes an adaptive route sampling framework that can flexibly return query results of different qualities according to the given response time limits;

- **Chapter 6** proposes bounded-cost informative route (BCIR) query, discusses its difference with existing route queries, and designs one exact solution with effective pruning methods and two approximate solutions with performance guarantees;

- **Chapter 7** defines probabilistic time-constrained route (PTR) query over uncertain road network models and designs a two-phase solution to efficiently process queries.

- **Chapter 8** concludes the thesis and highlights the future work relevant to road network and spatio-textual data.

Chapter 2

Literature Review

Spatial keyword has been studied for around ten years and a comprehensive survey on spatial keyword query could be found in [18]. According to the spatial aspect (i.e., the distance metric between two locations), exiting spatial keyword queries can be classified into two categories, i.e., ***spatial keyword queries in Euclidean space*** and ***spatial keyword queries over road networks***. For spatial keyword queries in Euclidean space, the spatial distance between two locations is Euclidean distance. In contrast, the spatial distance between two locations in spatial keyword queries over road networks is the network distance. The details of these two lines of research are elaborated in Sections 2.1 and 2.2, respectively. In addition, we also review the relevant work on route query over road networks since three of the four queries we will study are route queries.

2.1 Spatial Keyword Query in Euclidean Space

In the past decade, many spatial keyword queries in Euclidean space [18, 7] are proposed, which are elaborated as below.

- **Range Spatial Keyword Query** [20, 24, 42, 49, 80, 103]: Range spatial keyword query retrieves relevant spatio-textual objects within a given query range such that their textual descriptions contain query keywords or are relevant to

the query keywords in terms of some relevancy functions. In general, the query region is specified by a geometry shape, e.g., circle and square. The original solution [103] to range spatial keyword query first retrieves all the spatio-textual objects within the query range and then conducts a detailed examination on the retrieved spatio-textual objects based on their textual relevance to the query keywords. This solution is however inefficient for large-size datasets. To solve this inefficiency issue, previous studies try to embed traditional textual indices, e.g., Inverted Index [104] and Signature File [29] into an R-tree [40] or its variants. Almost all the proposed approaches for range spatial keyword query in Euclidean space are based on these hybrid indices. During the query processing, spatial proximity and textual relevance are computed simultaneously, thus making it efficient to prune irrelevant branches as soon as possible.

- **Boolean kNN Query** [12, 31, 85]: Given a query location and a set of query keywords, Boolean kNN query retrieves the k nearest spatio-textual objects that contain all the query keywords.

- **Top-k Spatial Keyword Query** [12, 25, 55, 70, 84]: Essentially, Boolean kNN query utilizes query keywords as a filter to find out those objects containing query keywords and then return the k nearest ones from the query location. Differently, top-k spatial keyword query combines spatial proximity and textual relevance with a score function and retrieves those spatio-textual objects having the top-k largest scores.

- **Group query** [4, 9, 10, 38, 58, 75, 98, 99, 101]: Instead of requiring spatol-textual objects to satisfy the query keywords requirement independently, group spatial keyword query aims to retrieve a set of spatio-textual objects to satisfy the query keywords requirement collectively.

- **Moving query** [45, 83, 86, 87, 89, 90]: For moving spatial keyword query, the query location is continuously moving and the corresponding query results also need to be updated frequently.

- **Why not query** [16, 17, 19, 57]: When issuing a spatial keyword query, users usually expect some known spatio-textual objects to be in the query results. In some cases, however, the real query results do not contain such objects and users would like to know the reason, thus the why not query. A why not query provides users the reason why the expected objects are missing from the query results and suggests a revised query to ensure that those objects are included.

- **Reverse kNN query** [23, 59]: For service providers, it is important to know the number of potential customers if they choose some business sites. A potential user here means his or her top-k query results for the same service provided by the service provider contain the service provider. Motivated by this observation, we have reverse kNN query. Given a query location and a set of query keywords, a reverse kNN query retrieves those objects whose top-k objects contain the query in terms of spatial proximity and textual relevancy.

- **Similarity join query** [5, 39, 93]: Given a set of spatio-textual objects and a set of query keywords, a similarity join query retrieves those pairs of objects such that the spatial proximity and relevancy between them satisfy some specified thresholds.

- **Region of interest query** [11, 21, 22, 32, 47, 79]: Given a set of query keywords, region of interest query aims to find a region (e.g., circle, rectangle and polygon) that is most relevant to the given query keywords.

2.2 Spatial Keyword Query over Road Networks

Compared with spatial keyword query in Euclidean space, the research on spatial keyword query over road networks is limited. On the one hand, the same query (e.g., top-k spatial keyword query) over road networks is much more complicated than that in Euclidean space because computing network distance requires a much higher computation cost than computing Euclidean distance. On the other hand, there are many new types of queries emerging when it comes to road networks. For example, it is of high interests for users to retrieve routes to cover some preferred query keywords.

Existing studies on spatial keyword queries over road networks are summarized as below.

- **Boolean kNN query:** Given a query location and a set of query keywords, Boolean kNN query over road networks retrieves the k nearest spatio-textual objects (based on network distance) that contain all the query keywords. Fang et al. [30] propose a solution with two pruning techniques to process Boolean kNN query over road networks. On the one hand, they employ an inverted index to conduct text pruning based the query keywords. On the other hand, a G-tree [102] is utilized to conduct the spatial pruning.

- **Top-k spatial keyword query:** Similar to the top-k spatial keyword query in Euclidean space, top-k spatial keyword query over road networks also targets at finding the k most optimal spatio-textual objects in terms of spatial proximity and textual relevancy. The only difference is that network distance is used for quantifying the spatial proximity instead of Euclidean distance. To solve this query problem, Rocha-Junior et al. [71] propose three approaches, i.e., 1) a basic approach based on road network expansion, 2) an enhanced approach

with indices to identify those spatio-textual objects relevant to query keywords, and 3) an overlay approach that partitions the road network into regions and constructs an hierarchy index for these regions. Particularly, the overlay approach can efficiently retrieve top-k spatio-textual objects by skipping those regions without relevant objects during the expansion.

- **Keyword-aware route query:** For spatial keyword query over road network, it is natural to combine route planning and spatial keyword query to retrieve the optimal routes for users. Cao et al. [8, 6] propose keyword-aware optimal route (KOR) query to find a route such that it covers all query keywords, its cost budget is less than a specified threshold, and its objective score is maximized. One example of KOR query is to find a route to covers "bank" and "restaurant" such that its distance is less than 5 km and its popularity is maximized. In KOR query, there are two types of weights, cost and score, for each route and KOR query constrains the cost by a cost budget and maximizes the score. Zeng et al. [97] assume that users have different preferences for different query keywords and propose to search for an optimal route that covers the weighted query keywords while satisfying a cover budget. To solve this problem, they propose a solution based on A* algorithm. Similar to [97], Li et al. [54] also consider different preferences for query keywords and propose Keyword-aware Dominant Routes (KDR) query to computes all the preferable routes that cover the query keywords while having a cost less than the cost budget. To solve KDR query problem, they develop both exact and heuristic algorithms. Yao et al. [95] propose approximate keyword route query to retrieve routes that cover the query keywords according to an approximate string similarity function rather perfect match. Another similar work is [74] which proposes the problem of identifying *Streets of Interest* (SOIs) aiming to identify individual

street segments that have a large density of relevant spatio-textual objects around them.

- **Group query:** We could also define group spatial keyword query over road networks to find a set of spatio-textual objects to collectively cover query keywords. To solve group keyword query problem over road network, Gao et al. [35] first prove that the query is NP-hard and then propose two approximate algorithms and one exact algorithm. Luo et al. [62, 63] develop a distributed solution to answering group spatial keyword queries over road networks. Concretely, they proposed a distributed index that enables each machine to independently evaluate the operation on its network fragment in a distributed setting. In addition, they theoretically prove the space optimality of the proposed index technique. As a variant of group spatial keyword query, Su et al. [77] propose to retrieve a set of objects for multiple users rather than a single one such that the returned objects are not only close to each other but also close to the group of users.

- **Region of interest query:** In some cases, users are interested in finding a region of interest. For example, it will be very helpful to get a region full of restaurants if a user wants to find a place for dining. To meet this demand, Cao et al. [11] propose the Length Constrained Maximum-Sum Region (LCMSR) query that computes a road network region such that its total length is less than a specified threshold and the spatio-textual objects in it best match the query keywords in terms of textual relevancy.

- **Moving query:** Guo et al. [37] consider the mobility of query locations and propose continuous top-k spatial keyword queries over road networks. To efficiently process the proposed query, they propose two algorithms, a query-

centric algorithm and an object-centric algorithm. The query-centric algorithm incrementally expands from the query location to search the k optimal spatio-textual objects. In contrast, object-centric algorithm expands from the spatio-textual objects relevant to query keywords directly.

- **Reverse kNN query:** Given a set of spatio-textual objects with both spatial locations and textual descriptions, a query location and a set of query keywords, a reverse kNN query over road networks retrieves those spatio-textual objects which have query as one of their top-k optimal objects in terms of both textual relevance and spatial proximity. With different ways to compute the top-k optimal objects for each object, we have different variants. For example, Gao et al. [34] compute the top-k optimal objects as the k nearest objects that contain all the query keywords and propose reverse top-k Boolean spatial keyword (RkBSK) query. To solve the RkBSK query, they devise an algorithm based on filter-and-refinement framework equipped with novel pruning techniques with respect to both spatial and textual information. Another example is the work of Luo et al. [61] who compute the top-k optimal objects based on a score function that combines both network distance and textual relevance. Accordingly, they propose reverse spatial and textual nearest neighbor (RSTkNN) query and design different pruning methods to facilitate the query processing.

- **Skyline route query:** Wen et al. [82] propose a variant of keyword-aware route query and try to compute the optimal routes in the similar way with skyline query. For a route in their query, they consider evaluation metrics including 1) the attractiveness of the spatio-textual objects the route passes, 2) the arrival times at these spatio-textual objects containing query keywords, and 3) the influence of those users who have passed the route. The three metrics aforementioned are then utilized for skyline search.

2.3 Route Query over Road Networks

Traditional shortest route query between two locations over road networks has been studied extensively and a comprehensive survey could be found in [88]. In addition to shortest route query between two locations, many practical route queries require returned routes to cover some specified objects, i.e., coverage route query. Roughly, coverage route query includes *arc orienteering problem* [60, 76, 81] *object coverage route query* [15, 52, 64, 73] and *keyword coverage route query* [8, 53, 54, 74, 95, 97].

- **Arc orienteering problem (AOP):** Arc orienteering problem [60, 76, 81] is a variant of the classical orienteering problem (OP). In orienteering problem, each network vertex has a score and each edge has a cost, and the objective is to find a route such that the cost is less than a given cost budget while the total score is maximized. Different from orienteering problem, the score in AOP problem is associated with edge instead of vertex. The objective of AOP problem is also to search a route such that its cost is within a cost budget and its total score is maximized.

- **Object coverage route query:** In this line of research, spatio-textual objects are grouped into different categories. The goal of object coverage route query is to find a route that covers at least one object from each required category of spatio-textual objects with the smallest travel distance. The required categories can be visited sequentially [73] or partially sequentially [15, 51]. In addition, Costa et al. [26] considers the service time of objects when searching the optimal route and Lian et al. [56] propose *trip planner query* which retrieves the optimal route to traverse a set of points over probabilistic time-dependent road networks.

- **Keyword coverage route query:** Keyword coverage route query retrieves

routes to cover user-specified query keywords. Cao et al. [8] propose the keyword-aware optimal route (KOR) query in which each edge of the road network was associated with a cost and a score. As discussed previously, given a cost budget, KOR query is targeted at finding a route that covers a set of user-specified query keywords and maximizes the objective score within the given cost budget. Taking into account the weights of different keywords of query objects, Zeng et al. [97] propose an optimization problem to optimize the keyword coverage on the required route. Yao et al. [95] proposed approximate keyword route query to retrieve routes that cover the query keywords according to an approximate string similarity function rather than perfect match. Li et al. [54] contribute an extension to the KOR query by searching all those routes that are not dominated by others. Another relevant work is [74] which proposes the problem of identifying *Streets of Interest* (SOIs) to identify individual street segments that have many relevant POIs around them.

In addition, considering the inherent uncertainty and dynamics of road networks, we have route queries over uncertain road networks. The uncertainty of travel times over road networks are generally represented in two forms, discrete probability distribution function and sample-based representation. With these two representations, we have different definitions for the optimality of a route over uncertain road networks. The existing studies on route queries over uncertain road networks can be classified into two categories, i.e., *least expected travel time route query* [41, 65, 13, 44, 14, 94] and *reliable route query* [3, 67, 68, 43, 72, 92, 91].

- **Least expected travel time route query:** Least expected travel route query retrieves the optimal route that has the minimum expected travel time. Both Hall [41] and Miller-hooks [65] utilize discrete probability distribution to represent the travel times of roads and compute the optimal route between two

locations based on the expected travel time. Chen et al. [14] employ a set of random travel time samples to capture the uncertainty of the travel time of roads and propose an efficient multi-criteria A* algorithm to exactly determine the least expected time route over uncertain time-dependent networks. Yang et al. [94] leverage a sampled-based representation scheme to construct an integer programming model for finding the a priori least expected time route. A moment-based characterization for continuous link travel times through variance is presented in [78] which computes the expected travel time for a given departure time with an ensemble mean travel time over a number of days. Chen et al. [13] propose three definitions of optimality, i.e., expected value model, dependent-chance model and chance-constrained model, for finding the optimal route under an uncertain environment. They designed a simulation-based generic algorithm to compute the final result. Assuming that link travel times are uncertain and correlated over time and space, Huang et al. [44] define a utility function of travel time to evaluate the routes and return the route with the minimum expected utility value as the query result.

- **Reliable route query:** Reliable route query aims to find the routes that can be travelled within certain time in high confidence. Hua et al. [43] apply random variables approximated by a set of samples to capture the uncertainty of road networks and propose three types of probabilistic queries, i.e., a probabilistic route query, a weighted-threshold top-k route query and a probability-threshold top-k route query. These queries can be computed with joint probability distribution directly. Wu et al. [67] study the problem of finding a priori optimal route such that a given likelihood of arriving on-time over uncertain networks is guaranteed. Xing et al. [92] use Lagrangian substitution approach to estimate the lower bound of the most reliable route solution through solving a sequence

of standard shortest route search problems. Zhou et al. [91] discuss two models to evaluate the travel time robustness, i.e., absolute and α-percentile robust shortest route problems. They propose a Lagrangian relaxation approach to deal with the problem.

2.4 Remarks

Table 2.1 summarizes the typical spatial keyword queries in Euclidean space and over road networks, where NA means inapplicability and $*$ indicates that there are no existing studies. Obviously, spatial keyword query in Euclidean space has been studied extensively. In contrast, the research on spatial keyword query over road networks is still limited and many query problems remain unsolved, e.g., range spatial keyword query, route query to search routes that are relevant to the query keywords, and keyword-aware route query over dynamic/uncertain road networks. Hence, in this topic "*spatial keyword query processing over road networks*", we study three typical types of spatial keyword queries to contribute to the literature on spatial keyword query over road networks.

Table 2.1: The typical spatial keyword queries in Euclidean space and over road networks.

Query	**Euclidean space**	**Road network**
range query	[20, 24, 42, 49, 80, 103]	*
Boolean kNN query	[12, 31, 85]	[30]
top-k query	[12, 25, 55, 70, 84]	[71]
group query	[4, 9, 10, 38, 58, 75, 98, 99, 101]	[35, 62, 63, 77]
moving query	[45, 83, 86, 87, 89, 90]	[37]
why not query	[16, 17, 19, 57]	*
reverse kNN query	[23, 59]	[34, 61]
similarity join query	[5, 39, 93]	*
region of interest query	[11, 21, 22, 32, 47, 79]	[11]
keyword-aware route query	NA	[8, 6, 97, 54, 95]
skyline route query	NA	[82]

To this end, however, we have to tackle multiple challenges. First, many spatial

keyword queries over road networks are NP-hard and it is difficult, if not impossible, to devise efficient exact solutions for these queries. Second, the size of road network could be very large, e.g., millions of vertices. In this case, much route computation required by spatial keyword queries will be time-consuming. Third, we still need to deal with the dual attributes, i.e., spatial and textual attributes, of spatial keyword query over road network. Therefore, this thesis aims to propose efficient solutions to the target queries under the challenges above.

Chapter 3

Preliminaries

Before proceeding to detail the techniques for processing the four target spatial keyword queries over road networks, we would like to briefly introduce the definitions of road networks and spatio-textual objects, several index structures that will be utilized for accelerating the query processing, and the datasets used for experimental evaluation.

3.1 Road Network and Spatio-Textual Objects

Definition 3.1 (Road Network)**.** *A road network is modeled as a directed graph* $G{=}(V, E)$*, where each vertex* $v_i{\in}V$ *represents an intersection of roads or a road terminal, and each edge* $e_{i,j}{\in}E(i{\neq}j)$ *represents the directed road segment from vertex* v_i *to vertex* v_j *and has a length of* $|e_{i,j}|$*.*

Figure 3.1(a) illustrates the road network of New York. A simple route (without duplicate vertices) R over road network G consists of a sequence of vertices, i.e., $R{=}(v_1, v_2, \ldots, v_\gamma)$ and $v_i{\neq}v_j$ if $i{\neq}j$ $(1 \leqslant i, j \leqslant \gamma)$. The length of R, $dist(R)$, is the sum of the lengths of its edges, i.e., $dist(R){=}\sum_{i=1}^{\gamma-1} |e_{i,i+1}|$. The shortest route from vertex v_i to vertex v_j is denoted by $SR(v_i, v_j)$, or $SR_{i,j}$ for simplicity.

Definition 3.2 (Spatio-Textual Objects)**.** *Each spatio-textual object, o, is denoted by a triple* $o{=}(l, e, \mathcal{K})$*, where:*

(a) Road network (NY) (b) Distribution of Objects (NY)

Figure 3.1: The road network and spatio-textual objects in New York.

- *$o.l$ is the spatial location consisting of longitude and latitude;*
- *$o.e$=(v_i, v_j) is the edge on which o resides or the nearest edge to o, and the distances from o to vertices v_i and v_j are $dist(o.l, v_i)$ and $dist(o.l, v_j)$, respectively;*
- *$o.\mathcal{K}$ is the textual description of o, e.g., the category, service contents and user comments.*

The set of all spatio-textual objects is denoted by $\mathcal{O}$.

Generally, the textual description $o.\mathcal{K}$ of o consists of a set of keywords. Figure 3.1(b) illustrates the spatial distribution of Points of Interest (POIs, a typical type of spatio-textual objects) in New York. Actually, there are many types of spatio-textual objects, e.g., POIs, geo-tagged comments and geo-tagged photos. In the context of road network, we mainly focus on POIs because spatial keyword queries over road networks usually need to find physical objects in the real world. Therefore, unless stated otherwise, spatio-textual object, object and POI are used interchangeably in this thesis.

3.2 Index Structures

3.2.1 R-tree

R-tree [40] is a popular spatial index that indexes spatial objects according to their spatial proximity. Figure 3.2 provides an example of R-tree which covers 9 objects. As illustrated by Figure 3.2, R-tree seeks to put close objects into the same tree node and organizes all nodes in a hierarchy structure.

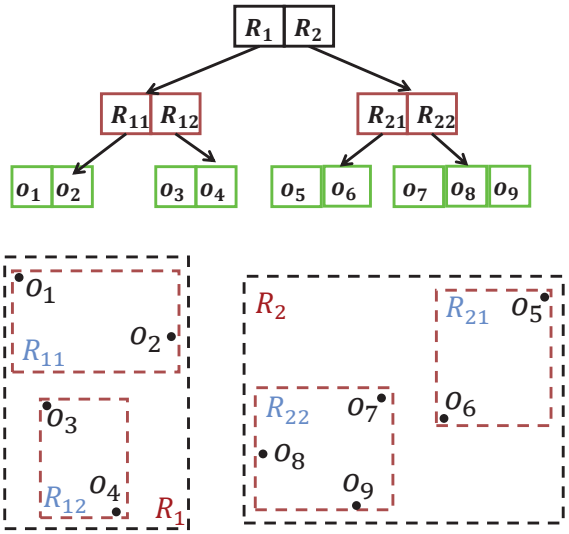


Figure 3.2: R-tree index.

3.2.2 Inverted Index

To efficiently retrieve the spatio-textual objects containing query keywords, Inverted Index [104] is widely used for indexing the text descriptions of spatio-textual objects. As illustrated in Table 3.1, Inverted Index lists for each keyword k_i all the objects containing k_i, denoted by $Inv(k_i)$. During the query processing, the spatio-textual objects containing query keywords can be quickly identified by using Inverted Index. For example, the set of spatio-textual objects containing keywords k_1 and k_2 can be obtained by computing the intersection of $Inv(k_1)$ and $Inv(k_2)$.

Table 3.1: Inverted Index for spatio-textual objects.

Keyword	**Spatio-textual object list**
k_1	o_1, o_3, o_5, $\cdots$
k_2	o_2, o_3, o_6, $\cdots$
$\cdots$	$\cdots$

3.2.3 Contraction Hierarchy

Contraction Hierarchy (CH) [36] is an efficient index for spatial network to compute the shortest route between locations. CH index imposes a total order on the vertices V according to their importance (e.g., the degree of vertex), and pre-computes the distances between vertices based on this order. Initially, the least important vertex, say v_i, is removed and its adjacent vertices are checked. If the shortest route of a pair of adjacent vertices, say v_j and v_k, passes through v_i, a virtual edge with the weight of the shortest distance between v_j and v_k is introduced. Then, the second least important vertex is removed and processed in the same way. By doing this repeatedly until the entire spatial network is reduced to an appropriate scale, we can build a hierarchical index for the spatial network. During the shortest distance/route computation, bidirectional Dijkstra's algorithm with some modifications, i.e., only those unvisited vertices (adjacent to current vertex) ranking higher (resp. lower) than the current expanding vertex are considered during the forward (resp. backward) traversal, is employed. Thus, much vertex expansion is avoided by using CH index, which greatly facilitates the query processing.

3.3 Datasets

In this thesis, we mainly employ two types of data, road network data and spatio-textual object data to evaluate the performance of proposed solutions. All the road network datasets and spatio-textual object datasets used in this thesis are summarized in Table 3.2.

Table 3.2: Datasets used for performance evaluation

Datasets	No. of vertices	No. of edges	No. of objects
New York (NY)	6, 393	13, 885	14, 537
Dublin (DL)	62, 975	82, 730	5, 297
Beijing1 (BJ1)	46, 029	62, 778	21, 192
Beijing2 (BJ2)	46, 029	62, 778	252, 125
California1 (CA1)	21, 048	21, 693	104, 470
California2 (CA2)	90, 870	202, 250	118, 307
Los Angeles (LA)	17, 458	40, 626	18, 384
London1 (LN1)	209, 406	282, 267	34, 341
London2 (LN2)	209, 045	282, 267	34, 341
Australia (AU)	1, 223, 171	1, 682, 182	70, 064
British Isles (BI)	3, 760, 213	4, 865, 094	300, 891

The details of these datasets are presented separately as below.

- **New York dataset**: New York dataset is directly extracted from OpenStreetMap[1](OSM). The road network and the distribution of spatio-textual objects in New York dataset are illustrated in Figure 3.1.

- **Dublin dataset**: The road network and spatio-textual objects (cf. Figure 3.3) in Dublin dataset are from [71] which also extracts them from OSM.

- **Beijing dataset**: The road network and spatio-textual objects in dataset BJ1 are extracted from OSM. Meanwhile, to get more spatio-textual objects, we also use another spatio-textual dataset from DataTang[2] in dataset BJ2. Figure 3.4 illustrates the road network and object distribution in dataset BJ1.

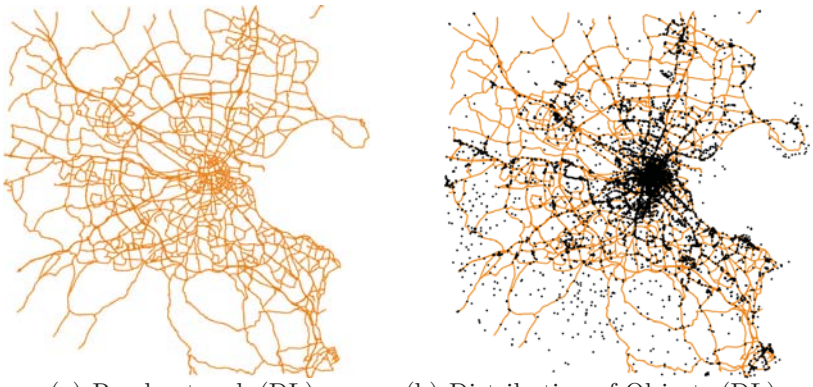
- **California dataset**: We have two datasets for California, i.e., CA1 and CA2, where CA1 is from *Real Datasets for Spatial Databases*[3] and CA2 is from OSM. Figure 3.5 shows the road network and the distribution of objects in CA2.

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

[2] http://www.datatang.com/

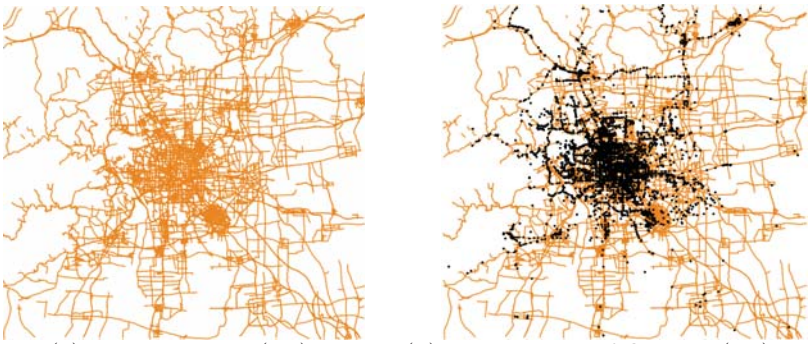
[3] http://www.cs.utah.edu/ lifeifei/SpatialDataset.htm

- **Los Angeles dataset**: Los Angeles dataset is extracted from OSM and its road network and spatio-textual objects are illustrated in Figure 3.6.
- **London dataset**: London dataset is from [71] where LN1 is the original dataset and LN2 removes some isolated vertices. Figure 3.7 illustrates the road network and the distribution of spatio-textual objects in LN1.
- **Australia dataset & British Isles dataset**: Both Australia dataset and British Isles dataset are extracted from OSM and their plots are illustrated in Figures 3.8 and 3.9, respectively.



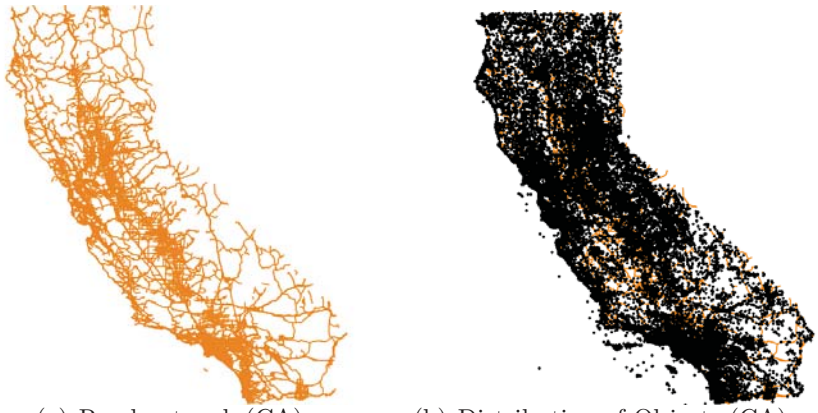
(a) Road network (DL) (b) Distribution of Objects (DL)

Figure 3.3: The road network and spatio-textual objects in Dublin.



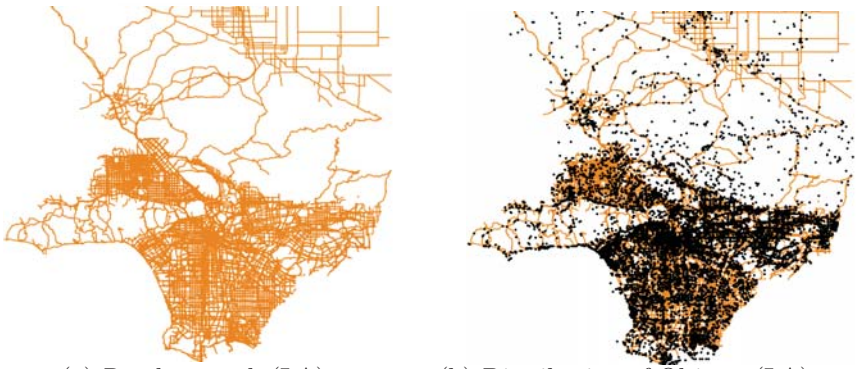
(a) Road network (BJ) (b) Distribution of Objects (BJ)

Figure 3.4: The road network and spatio-textual objects in Beijing.



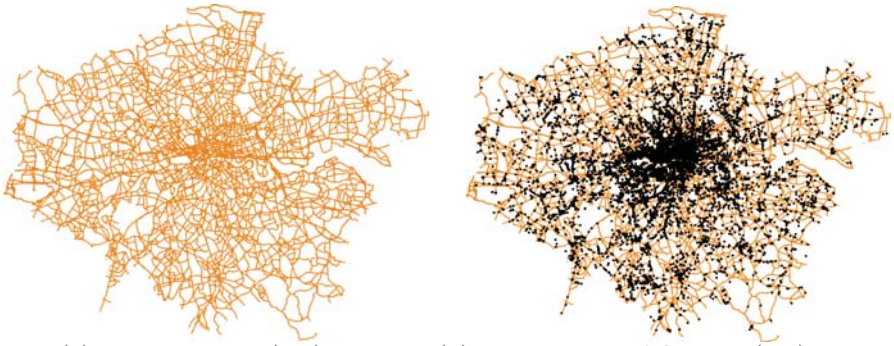
(a) Road network (CA) (b) Distribution of Objects (CA)

Figure 3.5: The road network and spatio-textual objects in California.



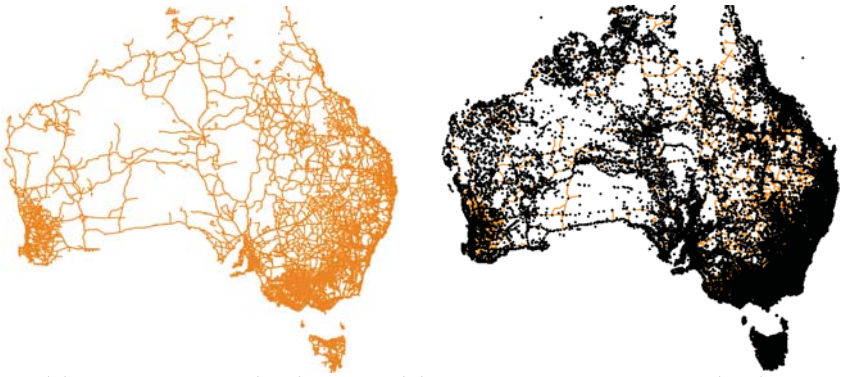
(a) Road network (LA) (b) Distribution of Objects (LA)

Figure 3.6: The road network and spatio-textual objects in Los Angeles.



(a) Road network (LN) (b) Distribution of Objects (LN)

Figure 3.7: The road network and spatio-textual objects in London.



(a) Road network (AU) (b) Distribution of Objects (AU)

Figure 3.8: The road network and spatio-textual objects in Australia.

(a) Road network (BI) (b) Distribution of Objects (BI)

Figure 3.9: The road network and spatio-textual objects in British.

3.4 Frequently used Notations

Table 3.3 lists the notations that will be frequently used in this thesis, where some notations are general for all the four queries while the others belong to specific queries.

Table 3.3: Frequently used notations and their meanings.

	Notation	Meaning
General	$G(V, E)$	a road network
	v_i	a vertex on road network G
	$e_{i,j}$	the edge from vertex v_i to vertex v_j
	$\|e_{i,j}\|$	the length of edge $e_{i,j}$
	R	a simple route
	$SR_{i,j}$	the shortest route from v_i to v_j
	$\|*\|$	the cardinality of set $*$
	$o = (l, e, \mathcal{K})$	a spatio-textual object
	$\mathcal{O}$	the whole set of spatio-textual objects
RSK	$q = (s, \mathcal{K}, \tau, r)$	a RSK query
	$\mathcal{O}_q$	returned objects for query q
KCR	$q = (s, d, \mathcal{K}, L)$	a KCR query
	$S(R)$	the number of query keywords covered by route R
BCIR	$q = (s, d, \mathcal{K}, B)$	a BCIR query
	R^*	the optimal route of BCIR query
	$\mathcal{K}_{i,j}$	the keywords of edge $e_{i,j}$
	$\mathcal{K}_R$	the keywords of route R
	$\mathcal{C}^B_{s,d}$	all the candidate routes from v_s to v_d
	$c_{i,j}$	the travel cost of edge $e_{i,j}$
	$f_{k,e}$	the occurrences of keyword k on edge e
	$f_{k,R}$	the occurrences of keyword k on route R
	$\lambda(R)$	the travel cost of route R
	$\tau(R)$	the score of route R in terms of textual relevancy
PTR	$q = (s, d, t_s, \mathcal{K}, h, \tau)$	a PTR query
	$\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}))$	uncertain road network
	$w(\mathcal{G})$	a possible world of $\mathcal{G}$
	$\mathcal{W}(\mathcal{G})$	all possible worlds of $\mathcal{G}$
	$\mathbf{t}(e_{i,j})$	the travel time of edge $e_{i,j}$
	$\mathbf{t}(R)$	the travel time of route R
	$o_i = \{l, e, \mathcal{K}, I\}$	a time constrained spatio-textual object
	$\mathbf{a}(o_i)$	the arrival time at object o_i
	$\mathbf{a}_c(o_i)$	the constrained arrival time at object o_i
	$R(s, d)$	a route from s to d covering required objects
	$\mathcal{R}^h_w(s, d)$	the top-h time-constrained routes in $w(\mathcal{G})$

Chapter 4

Range Spatial Keyword Query

4.1 Introduction

Given a query location and a set of query keywords, *Range Spatial Keyword* (RSK) query aims to find all the spatio-textual objects such that their textual descriptions are relevant to the query keywords while their locations are within a specified distance to the query location. For instance, a visitor may issue a RSK query to search for all the banks offering exchange service within 3 km from the current location. There have been some studies on RSK query in Euclidean space. In reality, however, people's move is generally constrained by the topology structure of road networks. Figure 4.1 illustrates an example of RSK query on a small road network with 13 vertices v_i (i=1, . . . , 13) and 10 spatio-textual objects o_j (j=1, . . . , 10). The number beside each edge is the length of the corresponding edge. The RSK query starting from s searches for all the banks within 3 km. In Euclidean space, both o_2 and o_4 will be returned as the query results. However, we cannot reach o_2 by travelling within 3 km from s along the road network since $dist(s, v_1) + dist(v_1, o_2)$=3.5 km. Hence, conducting RSK query based on network distance is more practical than that based on Euclidean distance.

Conducting RSK query on road networks is however much more challenging than that in Euclidean space because we need to evaluate the shortest distance between

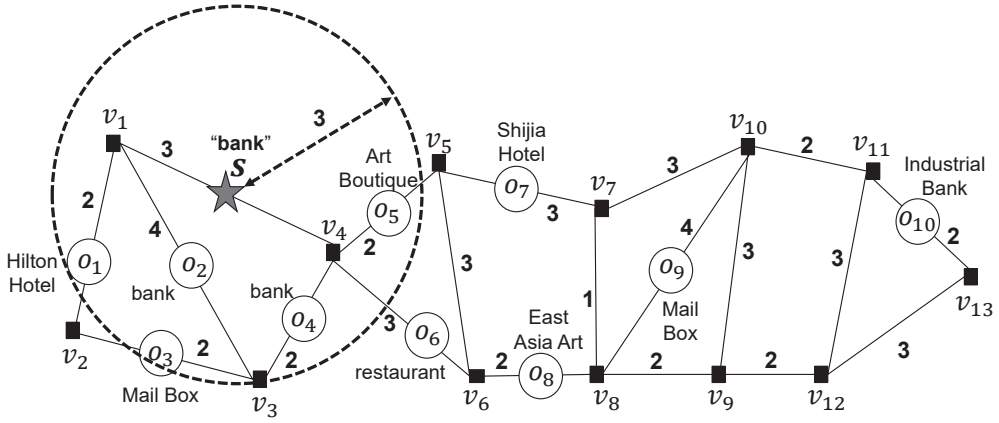


Figure 4.1: An example of *RSK* query, where $dist(s, v_1)$=1.5 km, $dist(v_1, o_2)$=2 km, $dist(s, v_4)$=1.5 km and $dist(v_4, o_4)$=1 km

query location and each candidate spatio-textual object. With a large query range, we have to enumerate many spatio-textual objects and compute their shortest distances to the query location, which incurs a high computational cost.

To efficiently solve RSK query problem over road networks, we propose three approaches:

- **Expansion-based approach** (EA): As a baseline approach, EA traverses the road network starting from the query location in a similar way to the classical Dijkstra's algorithm [28].

- **Euclidean heuristic approach** (EHA): EHA improves EA approach by employing Euclidean heuristic to accelerate the query processing.

- **Rnet Hierarchy based approach** (RHA): To avoid traversing the road network vertex by vertex, RHA leverages the Rnet Hierarchy [50] to partition the whole road network into a group of interconnected sub-networks and organizes them in a hierarchy index, which is then utilized for improving the query efficiency.

4.2 Problem Statement

With the definitions of road network and spatio-textual object in Definitions 3.1 and 3.2, we have the definition of RSK query over road networks as below.

Definition 4.1 (Range spatial keyword, RSK, query)**.** *A RSK query q is denoted by q=$(s, \mathcal{K}, \tau, r)$, where s is the query location, $\mathcal{K}$ is a set of query keywords, $\tau \in (0, 1]$ is a predefined textual relevance threshold, and r specifies the query range. Given road network G and spatio-textual objects $\mathcal{O}$, q aims to find the set of spatio-textual objects $\mathcal{O}_q \subseteq \mathcal{O}$ such that*

$$\mathcal{O}_q = \{o | (o \in \mathcal{O}) \wedge (dist(q.s, o.l) \leqslant r) \wedge (\theta(o.\mathcal{K}, q.\mathcal{K}) \geqslant \tau)\} \tag{4.1}$$

where $dist(q.s, o.l)$ is the network distance from $q.s$ to $o.l$, and $\theta(o.\mathcal{K}, q.\mathcal{K})$ computes the textual relevance between $o.\mathcal{K}$ and $q.\mathcal{K}$.

The textual relevance $\theta(o.\mathcal{K}, q.\mathcal{K})$ between $o.\mathcal{K}$ and $q.\mathcal{K}$ is computed by using the TF-IDF model [104], i.e.,

$$\theta(o.\mathcal{K}, q.\mathcal{K}) = \frac{\sum_{k \in q.\mathcal{K}} w_{k,o.\mathcal{K}} \cdot w_{k,q.\mathcal{K}}}{\sqrt{\sum_{k \in o.\mathcal{K}} (w_{k,o.\mathcal{K}})^2 \cdot \sum_{k \in q.\mathcal{K}} (w_{k,q.\mathcal{K}})^2}} \tag{4.2}$$

where $w_{k,o.\mathcal{K}}$=$1 + ln(f_{k,o.\mathcal{K}})$ with $f_{k,o.\mathcal{K}}$ counting the occurrences of keyword k in $o.\mathcal{K}$; $w_{k,q.\mathcal{K}}$=$ln(1 + \frac{|\mathcal{O}|}{|\mathcal{O}_k|})$ with $|\mathcal{O}|$ and $|\mathcal{O}_k|$ representing the number of spatio-textual objects on G and the number of spatio-textual objects containing k, respectively.

For ease of presentation, the following approaches are devised for undirected road networks, i.e., $|e_{i,j}|$=$|e_{j,i}|$ for edges $e_{i,j}$ and $e_{j,i}$. However, all approaches can be easily extended to directed road networks.

4.3 Expansion-based Approach

The basic idea of expansion-based approach (EA) is to expand vertex by vertex from query location until going beyond the query range. During the expansion, the spatio-

textual objects on edges are checked with respect to their textual relevance to the query keywords.

4.3.1 Index Structure

To facilitate the query processing using EA, we build index for both road network and spatio-textual objects. As illustrated in Figure 4.2, an R*-tree [2] (a variant of R-tree [40]) is employed to index edges E, where each edge is represented by a minimum bounding rectangle. With the R*-tree index, the edge on which query location s resides can be quickly determined with a spatial point query.

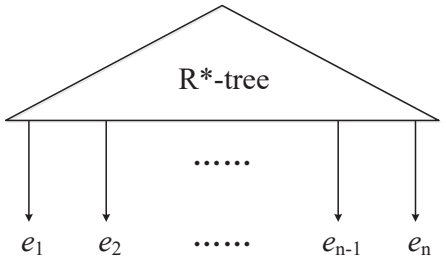


Figure 4.2: R*-tree for edges.

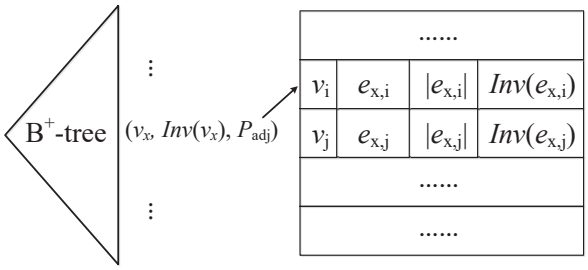


Figure 4.3: Index for road networks.

Meanwhile, as shown in Figure 4.3, we build a B^+-tree to index the modified adjacent list which maintains the connectivity of road network G. Each entry in leaf node is a triple $(v_x, Inv(v_x), P_{adj})$, where v_x is a vertex; $Inv(v_x)$ is a pointer to the inverted index [104] (called *vertex inverted index*) that indexes all the spatio-textual objects on v_x's adjacent edges; P_{adj} is another pointer pointing to the adjacent list of v_x. Each entry in the adjacent list is a quadruple $\langle v_i, e_{x,i}, |e_{x,i}|, Inv(e_{x,i})\rangle$, where v_i is a neighbor vertex of v_x; $e_{x,i}$ is the edge between v_x and v_i with length $|e_{x,i}|$; and $Inv(e_{x,i})$ is a pointer to the inverted index (called *edge inverted index*) that indexes the spatio-textual objects on edge $e_{x,i}$.

Updating the index structure efficiently is very important for practical use. In general, the topology structure of road network is comparatively static. Therefore, we focus on updating POIs. Once the description of POI o is updated, we need to

update the corresponding vertex inverted index and edge inverted index. Assuming that the total number of distinct keywords is η, the corresponding time complexity is $2 \cdot |o.\mathcal{K} \cdot |O(\eta)$, where $|o.\mathcal{K}|$ is the number of keywords in object o.

4.3.2 Query Processing

Initially, a priority queue U is created to store the visited vertices during expansion based on their network distances to the query location s. Meanwhile, a list L is created to store query results. First, the edge $e_{i,j}$ on which s resides is located by using the R*-tree for edges. Then, a verification is conducted to check whether $e_{i,j}$ contains spatio-textual object whose textual relevance to the query keywords $q.\mathcal{K}$ is larger than the threshold τ. Next, both vertices v_i and v_j are inserted into U with their distances to s. By obtaining vertices from U and checking their adjacent edges repeatedly, we can traverse all those edges within r from s and check them in the same way as we do for $e_{i,j}$. During the expansion, spatio-textual objects satisfying the textual relevance threshold are added to L. Finally, list L is returned as the query result.

Example 4.1. *We take the query q in Figure 4.1 for example, where query location is s, query keywords $q.\mathcal{K}$="bank" and query range r=3. For the simplicity of presentation, we ignore τ and only require that each returned spatio-textual object contains query keywords $q.\mathcal{K}$. First, we find that query location s is located on edge $e_{1,4}$ which has no desirable spatio-textual objects. Then, vertices v_1 and v_4 are inserted into queue U as $(v_1, 1.5)$ and $(v_4, 1.5)$, respectively. Next, we get $(v_1, 1.5)$ from U. By checking the inverted index $Inv(v_1)$ for vertex v_1, we find that edge $e_{1,3}$ contains query keyword "bank". Then, $Inv(e_{1,3})$ is checked and object o_2 containing query keyword "bank" is found. However, o_2 is not a desirable object because we have $dist(s, o_2)$=$dist(s, v_1) + dist(v_1, o_2)$=3.5>3. Since the distances from s to vertices v_2 and v_3 are larger than the query range r=3, we do not add these two vertices to*

queue U. Following that, we get $(v_4, 1.5)$ from U. Similarly, by checking the inverted index $Inv(v_4)$ of vertex v_4, we find that edge $e_{4,3}$ contains query keyword "bank". Then, $Inv(e_{4,3})$ is checked and the desirable object o_4 is inserted into list L. Since the distances from s to vertices v_3, v_5 and v_6 are larger than query range r=3, we do not search beyond them. Now queue U is empty and the algorithm terminates. Finally, we get the query result L=$\{o_4\}$.

4.3.3 Complexity analysis

EA approach expands from the query location s to check all the edges within query range r. Assuming that the number of vertices within query range is n' and the maximum degree of road network is D, the query complexity is $O(\log n + n' \cdot D \cdot F)$, where $\log n$ is the time cost for locating the edge of s with R*-tree, n is the total number of vertices, and F is the time cost to check one edge.

During the query processing, EA approach needs to store expanded vertices and obtained query results. Therefore, the space cost of EA is $O(n' + |L|)$, where L is the number of objects in the query results.

4.4 Euclidean Heuristic Approach

EA approach is efficient for RSK queries of small query ranges. However, if query range r is large and numerous edges and spatio-textual objects are covered, EA will incur a high overhead to obtain all the desirable objects because it needs to check the inverted indices of all the relevant edges that contain at least one query keyword. In general, many keywords only appear on a small number of edges. In this case, checking the inverted indices for all the relevant edges is unnecessary. To solve this issue, we propose the *Euclidean heuristic approach* (EHA).

Intuitively, the Euclidean distance between two vertices on a road network is always smaller, if not equal to, than the network distance between them. Therefore,

if a spatio-textual object belongs to the query result based on network distance, then it must be in the query result based on Euclidean distance. Motivated by this observation, we propose the EHA approach to first retrieve all the candidate objects using the state-of-the-art Euclidean distance-based approaches for RSK query. Here we employ the approach based on IR-tree[55] which augments each node of the R-tree with an inverted index. The set of edges containing candidate objects (satisfying textual relevance threshold) is recorded as E_q. During the expansion, we only check those edges in E_q. Thus, edges containing no desirable objects are filtered. EHA is implemented based on EA by adding a Euclidean distance-based RSK query at the beginning to get E_q.

4.4.1 Complexity analysis

Different from EA approach, EHA approach first uses IR-tree to retrieve all the objects whose Euclidean distances to the query location s are less than the query range r, and that satisfy the query keywords. The time complexity for this step is $O(\log|\mathcal{O}|)$, where $|\mathcal{O}|$ is the total number of objects. Then, EHA approach still needs to traverse all the edges whose network distance to s is less than r. Since we have known whether an edge contains required objects, we do not need to check the inverted files for all edges. The complexity for this traverse is $O(\log n+n'\cdot D\cdot\log|E_q|)$, where $\log|E_q|$ is used for checking whether one edge is in set E_q. The overall time complexity of EHA approach is thus $O(\log|\mathcal{O}|+\log n+n'\cdot D\cdot\log|E_q|)$.

As for space complexity, EHA approach needs to store all the edges E_q containing candidate objects. Therefore, the corresponding space complexity is $O(|E_q|+n'+|L|)$.

4.5 Rnet Hierarchy-based Approach

Essentially, both EA and EHA expand vertex by vertex on road network G until going beyond the query range r. Therefore, both approaches are inefficient to process

RSK queries with a large query range r. To solve this issue, we introduce the Rnet Hierarchy [50] to index road networks and further propose *Rnet Hierarchy-based approach*, RHA for short.

4.5.1 Indexing Structure

Rnet Hierarchy partitions a road network into sub-networks called Rnets (regional nets) and organizes them in a hierarchy structure as illustrated in Figure 4.4. An Rnet R is denoted by (V_R, E_R, B_R) where V_R, E_R, and B_R are the sets of vertices, edges and border vertices of R, respectively. B_R are those vertices shared by two or more Rnets (e.g., v_4). In Figure 4.4, there are four Rnets R_{11}, R_{12}, R_{21} and R_{22} on level 1 and two larger Rnets R_1 and R_2 on level 2. R_1 encloses R_{11} and R_{12}, and R_2 encloses R_{21} and R_{22}. Level 0 is the original road network. Therefore, a road network is organized as a group of Rnets on each level.

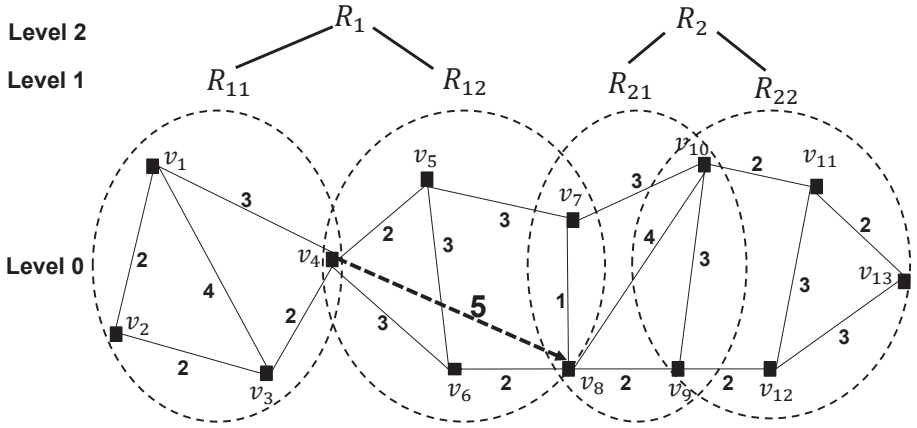
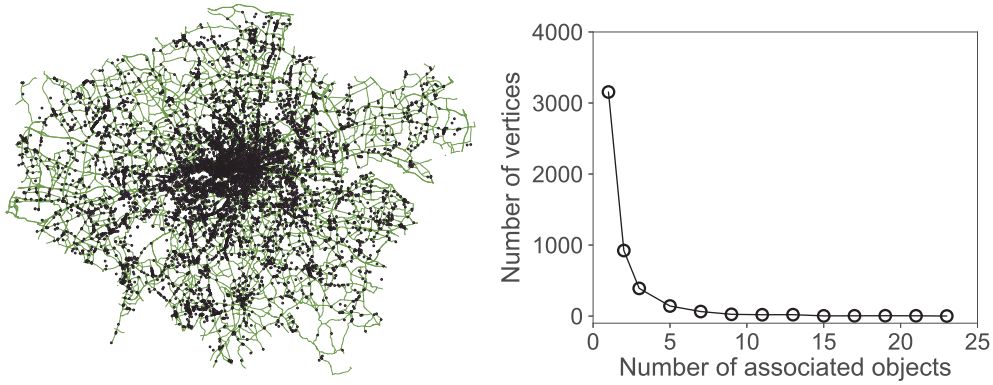


Figure 4.4: Rnet Hierarchy of the road network in Figure 4.1.

To skip over those Rnets containing no query keywords during query processing, *shortcut* is introduced. A *shortcut* is the shortest route between two border vertices of an Rnet, e.g., $SR(v_4, v_8)$. With the help of shortcuts on different levels, the search expands quickly with different step sizes. Now the challenge is how to partition a road network into a group of Rnets while minimize the number of border vertices. To

this end, we first consider the *equal-size partition* [50] using geometric approach [46] and *KL* algorithm [48] to partition the entire road network into Rnets of the similar size. Concretely, *Equal-size partition* first partitions the entire road network into two parts with almost the same number of edges, and then tunes them by exchanging edges to reduce the border vertices. By doing this recursively, we can partition G into a set of Rnets of the similar size.



(a) Spatial distribution of objects (b) Statistics of object distribution

Figure 4.5: The distribution of spatio-textual objects in London.

Equal-size partition ignores the spatial distribution of spatio-textual objects. In reality, however, spatio-textual objects are often clustered [96] in some hot areas like commercial centers. For example, the area around v_4 in Figure 4.1 has more objects than the areas around other vertices (e.g., v_{12}). Figures 4.5 illustrates distribution of spatio-textual objects in London. According to Figure 4.5(b), most vertices have less than 5 objects (those objects residing on the edges adjacent to the vertex). Accordingly, we consider partitioning a road network based on the distribution of objects, thus *distribution-aware partition*, and aim to partition as many objects as possible into the same Rnet. To this end, we first collect all the vertices with more objects than a specified parameter (e.g., 10) and then merge these vertices to form larger areas based on their spatial proximity until an Rnet is generated. Meanwhile, the other areas are partitioned by using the *equal-size partition*.

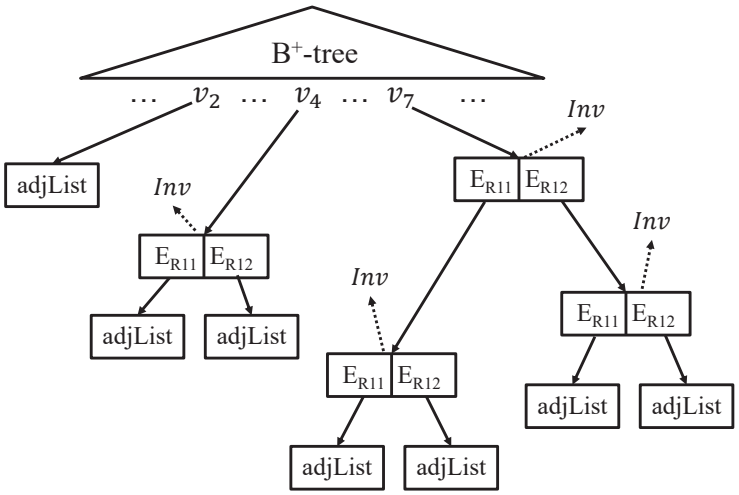


Figure 4.6: Index for the Rnet Hierarchy.

Rnet Hierarchy is organized in a B^+-tree as illustrated in Figure 4.6. The B^+-tree indexes all vertices based on their identifiers and each entry in the leaf node points to an adjacent list or a *hierarchy tree*. Concretely, if a vertex v_i (e.g., v_2) is not a border vertex, the entry for v_i has a pointer pointing to an adjacent list just as *EA*. Otherwise, the entry for v_i has a pointer pointing to a *hierarchy tree* which records the organization of all the Rnets associated with v_i on different levels. For example, v_4 is a border vertex of R_{11} and R_{12}, and it has a *hierarchy tree* T_{v_4}. The root node of T_{v_4} contains two entries E_{R11} (for R_{11}) and E_{R12} (for R_{12}) and a pointer pointing to the inverted index for them. In a *hierarchy tree*, each entry in the intermediate nodes also stores all the shortcuts within the corresponding Rnet while each entry in the leaf nodes points to the adjacent list of the border vertex.

In addition, to utilize Euclidean heuristic to quickly find out those Rnets containing desirable spatio-textual objects, we also organize all Rnets on different levels into a variant of IR-tree as illustrated in Figure 4.7.

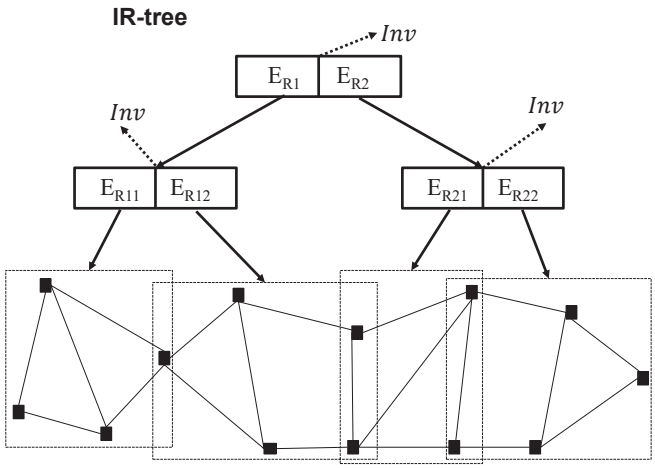


Figure 4.7: IR-tree for Rnets

4.5.2 Query Processing

Given an RSK query q, RHA searches for the desirable objects in an expanding fashion as detailed in Algorithm 4.1. If a vertex v_i is not a border vertex, it processes in the same way as EA approach. Otherwise, the hierarchy tree T_{v_i} of v_i is checked. First, we examine the inverted index for the root node of T_{v_i} to check whether it contains desirable objects. If not, we skip the entire Rnet via the shortcuts without checking its inner edges. Otherwise, we check its child nodes to further retrieve desirable objects.

Example 4.2. *We consider the RSK query in Figure 4.1 with query range r=3. First, we check the edge* $e_{1,4}$ *on which s resides. Then,* $(v_1, 1.5)$ *and* $(v_4, 1.5)$ *are inserted into queue U. Next, we get* $(v_1, 1.5)$ *from U. Since* v_1 *is not a border vertex, it is processed as the same as EA approach and no required objects are found. The, we get* $(v_4, 1.5)$ *from U. As* v_4 *is a border vertex, we evaluate* R_{11} *and* R_{12}*.* R_{11} *contains query keyword "*bank*" and a detailed examination is conducted. Then* $e_{4,3}$ *is checked and* o_4 *is added to L. We do not add* v_3 *to U since* $dist(s, v_3)$*=3.5 which is larger than the query range r.* R_{12} *contains no desirable objects and the shortcuts from* v_4 *to* v_7 *and* v_8 *go beyond the query range. Finally, the result* $L=\{o_4\}$ *is returned.*

Algorithm 4.1 The Rnet Hierarchy based Approach

Input: Road network $G = (V, E)$ and range query q=$(s, \mathcal{K}, \tau, r)$
Output: Any object o such that $dist(o.l, s) \leqslant r$ and $\theta(o.\mathcal{K}, q.\mathcal{K}) \geqslant \tau$

```
1: U ←newPriorityQueue()                    ▷ priority queue for storing visited vertices
2: L ←newList()                                          ▷ list to store final results
3: e_{i,j} ←locateEdge(s)
4: U.enqueue(v_i, dist(s, v_i))
5: U.enqueue(v_j, dist(s, v_j))
6: checkEdge(e_{i,j})                                          ▷ check Inv(e_{i,j})
7: while U is not empty do
8:     v ← U.Dequeue()
9:     if v is not a border vertex() then                    ▷ v is not a border vertex
10:        for each adjacent edge e of v do
11:            if e contains q.K then
12:                checkEdge(e)
13:            if v.currentDistance+e.length< r then
14:                U.enqueue(e.endVertex)
15:    else                                                  ▷ v is a border vertex
16:        check v.HierarcyTree
17: return L
```

Algorithm 4.1 can efficiently process RSK queries by skipping those regions without required spatio-textual objets. Therefore, RHA approach cost much less time than the previous two approached to evaluate the whole query range. As an expected improvement, we employ the IR-tree in Figure 4.7 to accelerate the query processing by computing all the Rnets containing desirable objects in advance, which avoids checking the inverted indices for all the Rnets.

4.5.3 Complexity Analysis

Since those regions without query results can be skipped, Algorithm 4.1 can get the query results efficiently. However, in the worst case (i.e., every edge contains at lease one candidate object), Algorithm 4.1 still needs to check the inverted file for every edge and the time complexity is the same as EA algorithm, i.e., $O(\log n + n' \cdot D \cdot F)$.

In the worst case, the space cost is $O(n' + |L|)$, where n' is the number of vertices within the query range, L is the number of objects in query results, which corresponds to U and L, respectively, in Algorithm 4.1.

4.6 Experiments

4.6.1 Setups

The performance of proposed approaches is evaluated on four real datasets[1], i.e., the road networks of Dublin (DL), London (LN), Australia (AU) and British Isles (BI). Table 4.1 shows the statistics of these datasets. For each dataset, we randomly generate 500 query locations within the road network area and 500 sets of keywords of sizes 1, 2, 3, 4, and 5, respectively. Table 4.2 lists the parameters used in the experiments and highlights their default values in bold. All experiments run on a PC with a 3.1 GHz Intel processor and a 4 GB RAM.

Table 4.1: Datasets for evaluating RSK query

Attributes	**Dublin**	**London**	**Australia**	**British Isles**
No. of vertices	62, 975	209, 406	1, 223, 171	3, 760, 213
No. of edges	82, 730	282, 267	1, 682, 182	4, 865, 094
No. of objects	5, 297	34, 341	70, 064	300, 891
No. of distinct keywords	3, 563	12, 522	18, 789	60, 558

Table 4.2: Parameter setting in RSK query

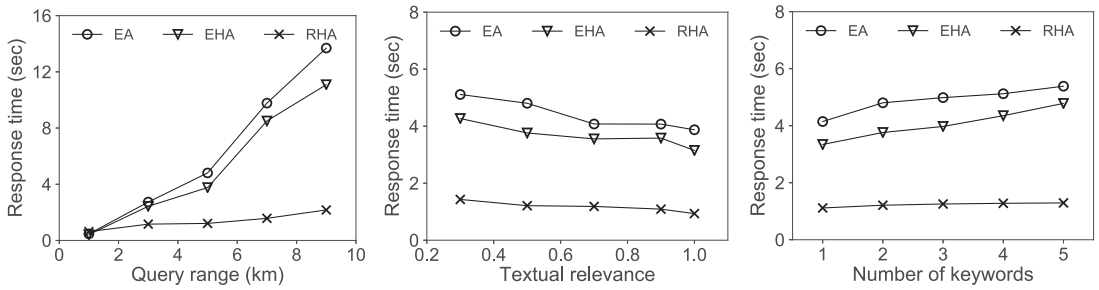
Parameters	**Values**
r	1, 3, **5**, 7, 9 (km)
τ	0.3, **0.5**, 0.7, 0.9, Boolean
$\|q.\mathcal{K}\|$	1, **2**, 3, 4, 5
Datasets	Dublin, London, Australia, **British Isles**
Partition Strategy	***Equal-size partition***, *Distribution-aware partition*

4.6.2 Experimental Results

Varying query range r: Figure 4.8(a) illustrates the response time while varying query range r. EA achieves satisfying performance for queries of small r. With the increase of r, however, the response time increases rapidly because EA has to check all the edges within r from s. Compared to EA, EHA performs better because it avoids checking the inverted indices for those edges without required spatio-textual objects.

[1] `http://www.idi.ntnu.no/~joao/publications/EDBT2012/`.

However, both EA and EHA expand vertex by vertex, which inevitably incurs a high overhead. In contrast, RHA performs better than EA and EHA because it bypasses the Rnets without required objects and avoids a detailed examination on their inner edges. Additionally, with the help of different layers and different size of Rnets, RHA adapts well to different r.



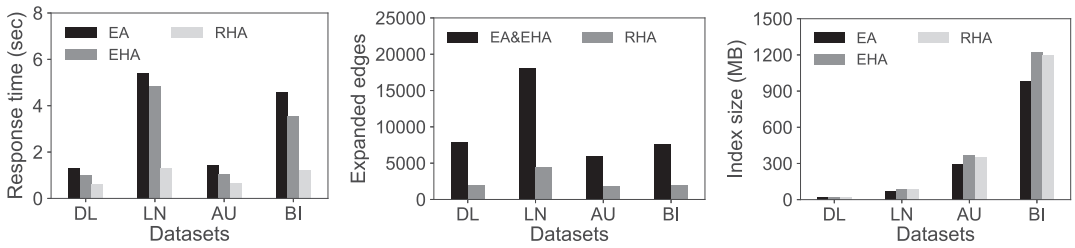
(a) Varying r (b) Varying τ (c) Varying $|q.\mathcal{K}|$

Figure 4.8: The response time while varying the query range r, the textual relevance threshold τ and the number of query keywords $|q.\mathcal{K}|$

Varying textual relevance threshold τ: Figure 4.8(b) shows the response time while varying the textual relevance threshold τ. A smaller τ covers more objects, thus consuming more time to evaluate these objects. Both EA and EHA cost more than 3 seconds to evaluate a RSK query when varying τ from 0.3 to 0.9. In contrast, RHA only needs about one second. As suggested by Figure 4.8(b), Boolean query[2] (τ=1) takes less time than other queries with textual relevance thresholds because it does not need to compute the textual relevance between spatio-textual objects and query keywords.

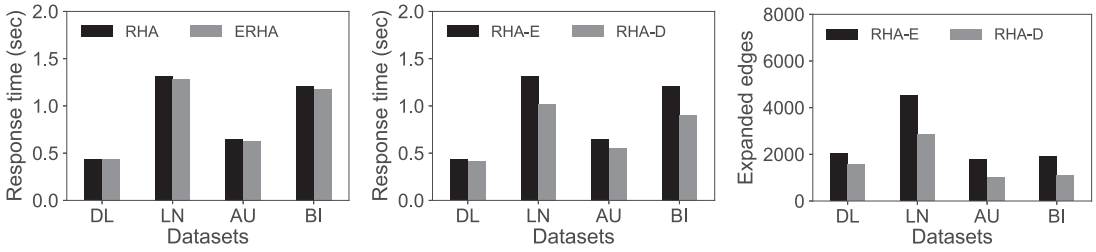
Varying number of keywords $|q.\mathcal{K}|$: Figure 4.8(c) presents the response time while varying the number of query keywords $|q.\mathcal{K}|$. With the increase of $|q.\mathcal{K}|$, both EA and EHA have to check more spatio-textual objects relevant to $q.\mathcal{K}$, thus leading to the increase in response time. Although RHA also needs more time to evaluates RSK queries of more query keywords, the increase of response time is moderate

[2] This value is only a label for Boolean query rather than a textual relevance threshold.



(a) Response time (b) Expanded edges (c) Index size

Figure 4.9: The response time, expanded edges and index size on different datasets



(a) ERHA (b) Response time (c) Expanded edges

Figure 4.10: The response time while using ERHA, and the response time and expanded edges while varying partition strategies

because RHA computes the textual relevance between an Rnet and query keywords $q.\mathcal{K}$ and does not examine the inner objects if the computed relevance is less than τ. Therefore, increasing the number of query keywords only affects a small portion of Rnets and the other Rnets can be still skipped through shortcuts.

Different datasets: Figure 4.9(a) shows the response time when using different datasets. RHA processes RSK queries on the four datasets of different sizes in about one second. Particularly, the response time on Australia network is smaller than that on London network though the Australia network is larger than London road network. This is because London network is denser than Australia network. Given the same query range, London network usually has more spatio-textual objects and edges than Australia network. This can be also seen from Figure 4.9(b), which illustrates the number of edges expanded during query processing.

Index size: Figure 4.9(c) shows the index size of the three approaches on different

datasets. The index size of EHA is larger than EA because it constructs an IR-tree for all objects. Meanwhile, RHA and EHA have almost the same index size.

Euclidean heuristic based RHA: Figure 4.10(a) presents the response time of the Euclidean heuristic based RHA (ERHA) that indexes all Rnets with an IR-tree as illustrated in Figure 4.7. As RHA indexes a road network in a hierarchy and is already able to prune unrelated Rnets, there is no obvious difference between RHA and ERHA with respect to the response time.

Partition strategy: Figures 4.10(b) and 4.10(c) present the response time and the number of expanded edges while using different road network partition strategies. Compared with *equal-size partition* (RHA-E), *distribution-aware partition* (RHA-D) achieves better performance. By partitioning a road network based on the distribution of objects, the areas containing few objects can be partitioned into Rnets of large granularity, thus making RHA-D more advantageous in pruning unrelated Rnets than RHA-E.

4.7 Summary

In location-aware applications, RSK query has been widely used for retrieving nearby services for users. Considering that existing techniques for RSK query cannot be applied to its counterpart over road networks, we thus propose RSK query problem over road networks. To efficiently process this query, we first propose the EA approach based on the locality of RSK query. Then, EA approach is improved by utilizing the observation that network distance is always larger than or equal to the Euclidean distance between two locations, thus EHA approach. In addition, to process RSK queries over large road networks and RSK queries of large query range, we design Rnet Hierarchy index and propose the RHA approach which can efficiently process RSK queries of different query ranges over road networks of millions of vertices.

Chapter 5

Keyword Coverage Route Query

5.1 Introduction

In some cases, users have the start location and destination, and wish to explore some interested spatio-textual objects along the routes. Existing route queries aim to find the shortest route that covers spatio-textual objects of specific categories [52, 73] or spatio-textual objects containing the query keywords [8, 6]. One example of these queries is "*finding the shortest route that passes through a restaurant and a pub*". Such queries are useful when users need to visit each category of spatio-textual objects once. In contrast, we focus on a different case where the user wants to explore as many relevant spatio-textual objects as possible. For example, a tourist may ask: "*finding a route that covers as many restaurants and pubs as possible*". This would provide more choices for users to choose, and thus improve the satisfaction of users.

Therefore, we propose *keyword coverage route (KCR) query* in this chapter. Given the user's interests (i.e., query keywords) and travel requirements (i.e., start location s, destination d, and route distance threshold), KCR query returns the optimal route from s to d such that: the length of the route is bounded by the given distance threshold; and the total number of query keywords on the route is maximized. The length is restricted because users may not want to travel a very long route. Essential-

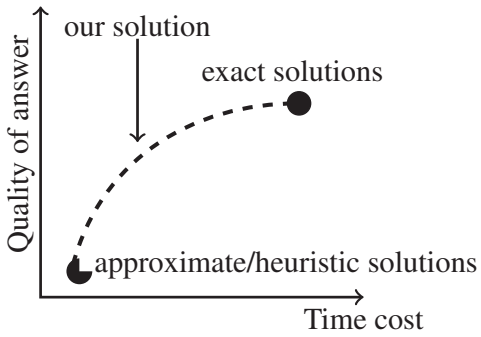


Figure 5.1: The quality of query result and the response time of exact and approximate/heuristic solutions

ly, KCR query aims to find a route that covers the most number of query keywords while satisfying the given distance threshold.

We will prove that KCR query is NP-hard (cf. Section 5.2.2). For such a computationally expensive problem, the typical solutions are either to develop an exact solution (e.g., branch-and-bound solution) that computes the optimal result but incurs a significant time cost, or to develop an approximate solution (e.g., nearest neighbor heuristics) that quickly returns results of low quality. Figure 5.1 visualizes the relationship between the quality of query result and the response time for these two types of solutions. In this work, we propose an adaptive solution that provides flexible trade-offs between the quality of query result and response time. Concretely, we take the response time limit into account and propose an adaptive route sampling framework to compute a good approximate route within the given response time limit.

5.2 Problem Statement

5.2.1 Problem Definition

Based on the definitions of road network and spatio-textual objects in Section 3.1, we have the definition of *keyword coverage route (KCR)* query as below.

Definition 5.1 (Keyword Coverage Route (KCR) Query)**.** *Given a road network G and the spatio-textual objects $\mathcal{O}$, a keyword coverage route query is denoted by $q = (s, d, \mathcal{K}, L)$, where s and d are the start location and destination, respectively, $q.\mathcal{K}$ is a set of query keywords to describe uers' interests, and L is a distance threshold. The objective of q is to find a route R^* from s to d such that:*

$$R^* = \arg \max_{R \in \mathcal{R}_L(s,d)} S(R) \tag{5.1}$$

where

- *$\mathcal{R}_L(s, d)$ is the set of feasible routes between s and d whose lengths are less than L, i.e., $\mathcal{R}_L(s, d)$=$\{R | R \in \mathcal{R}(s, d) \wedge dist(R) \leqslant L\}$ and $\mathcal{R}(s, d)$ represents all the simple routes from s to d;*

- *$S(R)$ is the score of R with respect to the query keywords $q.\mathcal{K}$ and computed by $S(R)$=$\Sigma_{o_i \in R} |o_i.\mathcal{K} \cap q.\mathcal{K}|$, where $|o_i.\mathcal{K} \cap q.\mathcal{K}|$ computes the number of query keywords contained by $o_i.\mathcal{K}$ and is denoted by $S(o_i)$.*

In the definition above, we say $o \in R$ if $o.e$ is on route R. Meanwhile, we have the score of an edge $S(e)$=$\Sigma_{o_i \in e} S(o_i)$. Accordingly, we have $S(R)$=$\Sigma_{e \in R} S(e)$.

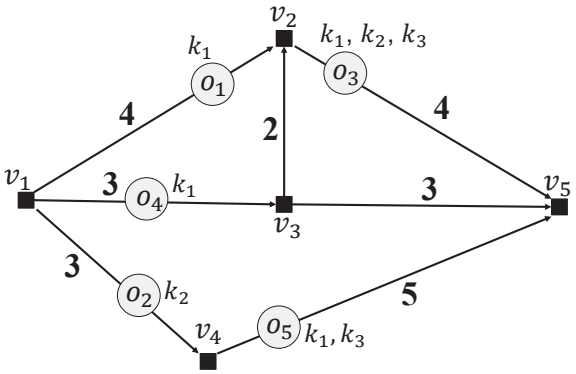


Figure 5.2: Road network and KCR query q=$(v_1(s), v_5(d), \{k_1, k_2\}, 8)$

Table 5.1: Routes for the KCR query in Figure 1

Routes	**Length**	**Objects on route**	**Score**
R_1=$\langle v_1, v_3, v_5\rangle$	6	$\{o_4\}$	1
R_2=$\langle v_1, v_2, v_5\rangle$	8	$\{o_1,o_3\}$	3
R_3=$\langle v_1, v_3, v_2, v_5\rangle$	9	$\{o_3,o_4\}$	3
R_4=$\langle v_1, v_4, v_5\rangle$	8	$\{o_2,o_5\}$	2

Example 5.1. *Figure 5.2 illustrates an example of road network with five vertices and five spatio-textual objects. The length of each edge is marked beside the corresponding edge. Figure 5.2 also provides a KCR query, where s=v_1 and d=v_5 are the start location and destination, respectively, $q.\mathcal{K}$=$\{k_1, k_2\}$ specifies two query keywords, and the distance threshold is L=8. We have four possible routes as listed in Table 5.1. Route R_3 should be pruned because it is not a feasible route ($dist(R_3)$=9>8). The other three routes are feasible because all of them satisfy the distance threshold. Finally, the query result is R_2 since its score is larger than that of routes R_1 and R_4.*

5.2.2 Problem Hardness

In complexity theory, a problem is NP-hard if we can prove its corresponding decision problem is NP-hard. Therefore, we first define the decision version of KCR query problem, i.e., *Decision-KCR*, as below.

Definition 5.2 (Decision-KCR problem)**.** *Given road network G, Decision-KCR problem decides whether there exists a route from start location s to destination d such that its length is at most L and its score is at least X, where $X \geqslant 0$.*

Then, we have the following theorem.

Theorem 5.1. *Decision-KCR problem is NP-hard.*

Proof. The theorem can be proved by a reduction from the Hamiltonian Path/Route (Ham-Route) problem, a well-known NP-hard problem, to the Decision-KCR problem. Given a graph $G(V, E)$, a Ham-Route problem decides whether there exists a simple route on G that visits every vertex in V once.

Assume that graph $G(V, E)$ is an instance of Ham-Route problem and that graph $G'(V', E')$ is an instance of Decision-KCR problem. First, we convert G to G'. Initially, we let G' have the same vertices of G, i.e., $V{=}V'$. If $e_{i,j}{\in}E$, we set the length and score of edge $e_{i,j}$ in G' as $|e_{i,j}|{=}1$ and $S(e_{i,j}){=}1$. Otherwise, we have $|e_{i,j}|{=}+\infty$ and $S(e_{i,j}){=}0$. Two additional vertices $v_s{=}s$ and $v_d{=}d$ are added to G' such that, for any vertex $v_i{\in}V'$, we have $|e_{s,i}|{=}|e_{d,i}|{=}1$ and $S(e_{s,i}){=}S(e_{d,i}){=}0$. Now we have $V'{=}V \cup \{v_s, v_d\}$.

We can prove that a Ham-Route problem on graph G is equivalent to a Decision-KCR problem on graph G'. Specifically, G has a Ham-Route if and only if there is a route in G' from v_s to v_d such that the length of the route is at most $n + 1(n{=}|V|)$ and the score of the route is at least $n - 1$, which can be proved from the following two aspects.

- If there is a Ham-Route R in G, the number of vertices in R should be n. By adding v_s and v_d to the two ends of R, respectively, we can generate a new route R' whose length and score are $n + 1$ and $n - 1$, respectively.

- If there is a route R' in G' such that $dist(R'){\leqslant}n + 1$ and $S(R'){\geqslant}n - 1$, the number of edges in R' (except for the edges connected to v_s and v_d) should be at least $n - 1$ because the score of each edge is at most 1. Therefore, the total number of edges in R' is at least $n + 1$ if the two edges connected to v_s and v_d are counted. Considering that the length of R' is at most $n + 1$ and the length of each edge is 1, we can conclude that the number of edges in R' is $n + 1$ and the number of vertices in R' is $n + 2$. Thus, a Ham-Route in G is obtained by removing v_s and v_d from R'.

The proof is completed. □

5.3 Adaptive Route Sampling Framework

5.3.1 Framework Overview

As KCR query problem is NP-hard, it is impossible to design algorithms with polynomial time complexity when P≠NP. Therefore, it is reasonable to design approximate solutions for KCR queries. Meanwhile, to overcome the low efficiency of exact solutions and the low quality of approximate/heuristic solutions, we propose a route sampling framework, as illustrated in Figure 5.3, which adaptively generates query results of different qualities according to the given response time. Taking as input the road network and a KCR query, the sample route module generates a feasible route with a certain route sampling method each time. The current optimal route is updated if the new feasible route has a larger score. Then, the response time limit is checked. If there is still spare time, more loops are executed to generate new feasible routes. Otherwise, the current optimal route is returned as the query result.

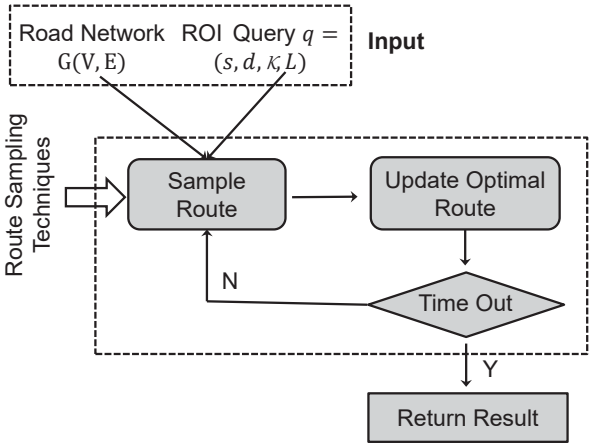


Figure 5.3: The adaptive route sampling framework for KCR query

This route sampling framework concerns two key issues, i.e., sampling a feasible (i.e., the feasibility issue) and good (i.e., the quality issue) route each time. We will elaborate these two issues in the following sub-sections.

5.3.2 Sampling a Feasible Route

We sample a feasible route R by starting from s and adding edges sequentially until reaching the destination d. The challenging issue is how to ensure that R is feasible. Without loss of generality, we assume that the current expanding vertex is v_i with adjacent vertices $adj(v_i)$, i.e., $adj(v_i)=\{v_j|e_{i,j} \in E\}$. After removing the visited vertices, we get the unvisited adjacent vertices of v_i, i.e., $adj_u(v_i)$. Then, a vertex v_j is selected from $adj_u(v)$ according to selection probability $p_{i,j}$ and concatenated to R to generate a longer partial route, where $0 \leqslant p_{i,j} \leqslant 1$ and $\sum_{v_j \in adj_u(v_i)} p_{i,j}=1$. The details of $p_{i,j}$ will be elaborated in Section 5.3.3. However, some vertices from $adj_u(v)$ may violate the distance threshold. In order to guarantee a feasible route, the vertex v_j selected from $adj_u(v)$ must survive from the distance pruning, i.e., the following lemma.

Lemma 5.1. *For any vertex $v_j \in adj_u(v_i)$, v_j can be selected for route expansion if and only if the following condition holds:*

$$dist(R) + |e_{i,j}| + dist(SR(v_j, d)) \leqslant L \tag{5.2}$$

where R is the partial route from start location s to vertex v_i, $|e_{i,j}|$ is the length of the edge from v_i to v_j, and $dist(SR(v_j, d))$ is the shortest distance between v_j and the destination d.

Proof. The proof is obvious because any vertex $v_j \in adj_u(v_i)$ satisfying Eq. (5.2) can generate a feasible route. □

We record those vertices in $adj_u(v_i)$ surviving from the distance pruning of Lemma 5.1 as $adj'_u(v_i)$. Therefore, the selection of vertices to expand the route should be constrained by the distance threshold L and we call it *selection constrained* (SC) expansion. Then, we have the following theorem.

Theorem 5.2. *SC expansion can always generate a feasible route.*

Proof. Assume the current partial route and the newly added vertex are R and v_i, respectively. Initially, $R{=}\langle s\rangle$ and $v_i{=}s$. In each expansion, an adjacent vertex v_j satisfying Eq. (5.2) is selected. We can always generate a feasible route $R'{=}R \oplus \{v_j\} \oplus SR(v_j, d)$ because it satisfies the distance threshold L, where $x \oplus y$ represents a concatenation operation between x and y. □

Example 5.2. *We take the KCR query in Figure 5.2 for example. Initially, we have* $R{=}\langle v_1\rangle$ *and* $adj'_u(v_1){=}\{v_2, v_3, v_4\}$. *Assuming that* v_3 *is selected and* $R{=}\langle v_1, v_3\rangle$, *we have* $adj_u(v_3){=}\{v_2, v_5\}$. *Since*

$$dist(R) + |e_{3,2}| + dist(SR(v_2, d)) = 3 + 2 + 4 = 9 > L = 8,$$

we have $adj'_u(v_3){=}\{v_5\}$. *Finally, a feasible route* $R{=}\langle v_1, v_3, v_5\rangle$ *is sampled.*

In SC expansion, $dist(SR(v_j, d))$ is computed online. To accelerate this computation, we conduct a backward expansion from the destination d on the reverse graph of G (generated by changing the direction of each edge in G) to compute the shortest distances from d to all the vertices that are within L from d. These distances are stored and can be checked quickly during the SC expansion, thus avoiding computing the network distance between v_j and d for every partial route.

The process of feasible route sampling is outlined in Algorithm 5.1, which receives as input a KCR query q and returns as output a feasible route R.

5.3.3 Route Sampling Techniques

Though SC expansion guarantees a feasible route, how to select a vertex from $adj'_u(v_i)$ is not yet tackled because the selection probability $p_{i,j}$ is still unknown. The quality issue depends on the techniques used to compute $p_{i,j}$. In the following subsections, we will elaborate different techniques to compute $p_{i,j}$. Note that, other route sampling techniques can also be added to the framework.

Algorithm 5.1 Sample a feasible route

Input: a *KCR* query $q = (s, d, \mathcal{K}, L)$
Output: a feasible route R

```
1: Initialize route R ← ⟨s⟩ and v_i ← s
2: while R not reach d do
3:     adj'_u(v_i) ← ∅
4:     for v_j ∈ adj(v_i) do
5:         R' ← R ⊕ {v_j} ⊕ SR(v_j, d)
6:         if v_j ∉ R ∧ dist(R') ⩽ L then
7:             adj'_u(v_i).add(v_j)
8:     let p_{i,j} be the probability to select vertex v_j
9:     pick v_j from adj'_u(v_i) according to p_{i,j}
10:    R.add(v_j) and v_i ← v_j
11: return R
```

A. Uniform Route Sampling

Straightforwardly, during the expansion, a vertex can be selected from $adj'_u(v_i)$ uniformly, i.e., $p_{i,j}=\frac{1}{|adj'_u(v_i)|}$. By doing this repeatedly until reaching the destination d, we can generate a feasible route.

Example 5.3. *We take the KCR query in Figure 5.2 for example and expand from* v_1 *with the partial route* $R=\langle v_1\rangle$. *Initially, by Lemma 5.1, we have* $adj'_u(v_1)=\{v_2, v_3, v_4\}$. *A vertex is then selected from* $adj'_u(v_1)$ *randomly with equal probability. We assume that* v_2 *is selected and* $R=\langle v_1, v_2\rangle$. *Next, we have* $adj_u(v_2)=\{v_3, v_5\}$. *However,* v_3 *is removed because* $dist(R)+|e_{3,2}|+dist(SR(v_2, d)) = 3+2+4 = 9 > L = 8$. *Therefore, only one vertex is left, i.e.,* $adj'_u(v_2)=\{v_5\}$. *Now, the final vertex* v_5 *is reached and we get the feasible route* $R=\langle v_1, v_2, v_5\rangle$ *with a score* $S(R)=S(e_{1,2}) + S(e_{2,5})=3$.

B. Prioritized Route Sampling

All vertices in $adj'_u(v_i)$ are selected equally in the uniform route sampling. Therefore, in most cases, we cannot obtain a good route unless enough feasible routes are sampled. Intuitively, a good route sampling method should give a higher priority to those routes with large scores. However, it is difficult to achieve this end because we do not have the whole route before finishing the route sampling. In other words, we

only have a partial route when selecting a vertex from $adj'_u(v_i)$. Actually, selecting v_j from $adj'_u(v_i)$ can be done in a greedy fashion. To this end, we define the desirability of selecting a vertex $v_j \in adj'_u(v_i)$ as below.

$$\eta_{i,j} = 1 + \frac{S(e_{i,j})}{|e_{i,j}| \cdot S^+_{avg}(G)} \tag{5.3}$$

where $S^+_{avg}(G)$ is the maximum average score over the whole road network, i.e.,

$$S^+_{avg}(G) = \max_{e \in E} S_{avg}(e) = \max_{e \in E} \frac{S(e)}{|e|} \tag{5.4}$$

We compute the selection probability $p_{i,j}$ of v_j by $p_{i,j}=\frac{\eta_{i,j}}{\sum_{v_h \in adj'_u(v_i)} \eta_{i,h}}$. In this way, the scores of adjacent edges are taken into consideration when selecting the next vertex to expand. Though this is just a local greedy strategy, it indeed improves the quality of returned routes compared with the uniform route sampling, which will be validated by the experiments.

Example 5.4. *We still take the KCR query in Figure 5.2 for example. First, we have* $S(e_{1,2})= S(e_{1,3})= S(e_{1,4})=S(e_{4,5})=1$, $S(e_{2,3})=S(e_{3,5})=0$, *and* $S(e_{2,5})=2$. *By Eq. (5.4), we have* $S^+_{avg}(G)=0.5$. *Initially, we start from* v_1 *with route* $R=\langle v_1 \rangle$ *and have* $adj'_u(v_1)=\{v_2, v_3, v_4\}$ *by Lemma 5.1. Particularly, for vertex* v_2, *we have*

$$\eta_{1,2} = 1 + \frac{S(e_{1,2})}{|e_{1,2}| \cdot S^+_{avg}(G)} = 1 + \frac{1}{4 * 0.5} = 1.5$$

In such a way, we have $\eta_{1,3} = \eta_{1,4} = 1.67$. *Further, we have*

$$p_{1,2} = \frac{\eta_{1,2}}{\sum_{v_h \in adj'_u(v_1)} \eta_{i,h}} = \frac{1.5}{1.5 + 1.67 + 1.67} = 0.31.$$

Similarly, we have $p_{1,3}=p_{1,4}=0.345$. *A vertex is then selected from* $adj'_u(v_1)$ *according to the computed probability* $p_{i,j}$. *Without loss of generality, we assume that* v_2 *is*

selected and R=$\langle v_1, v_2\rangle$. *Next, we have* $adj_u(v_2)$=$\{v_3, v_5\}$. *However,* v_3 *is removed due to the distance violation as discussed previously. Therefore, only one vertex is left, i.e.,* $adj'_u(v_2)$=$\{v_5\}$ *and we have* $p_{2,5}$=1. *Now, we reach the final vertex* v_5 *and have a feasible route* R=$\langle v_1, v_2, v_5\rangle$ *whose score is* $S(P)$=$S(e_{1,2})$ + $S(e_{2,5})$= 3.

C. Dynamic Prioritized Route Sampling

To take advantage of the previously sampled routes, we further propose the dynamic prioritized route sampling technique, which updates the selection probability $p_{i,j}$ dynamically according to the knowledge learned from the sampled routes. Assuming that a set of routes have been sampled by using the prioritized route sampling, different strategies to update $p_{i,j}$ are elaborated below.

DPri-Sample 1: Intuitively, the probability of getting better routes increases as more and more routes are sampled. Therefore, for each edge $e_{i,j} \in R$, we update its desirability by $\eta'_{i,j}$=$\frac{\eta_{i,j}}{1+\mathcal{R}(e_{i,j})}$, where $\mathcal{R}(e_{i,j})$ is the number of sampled routes that pass edge $e_{i,j}$. Accordingly, $p_{i,j}$ is adjusted as below.

$$p_{i,j} = \frac{\eta_{i,j}/(1+\mathcal{R}(e_{i,j}))}{\Sigma_{v_h \in adj'_u(v_i)}\{\eta_{i,h}/(1+\mathcal{R}(e_{i,h}))\}} \tag{5.5}$$

DPri-Sample 2: Intuitively, we tend to travel these edges covered by routes of high scores. Therefore, we increase the desirability of an edge $e_{i,j}$ with respect to the scores of the routes that travel through $e_{i,j}$, i.e.,

$$p_{i,j} = \frac{\eta_{i,j} \cdot (1+\Sigma_{k=1}^{m}\Delta\tau_{i,j}^{k})}{\Sigma_{v_h \in adj'_u(v_i)}\eta_{i,h} \cdot (1+\Sigma_{k=1}^{m}\Delta\tau_{i,h}^{k})} \tag{5.6}$$

where $\Delta\tau_{i,j}^{k}$ is the increase ratio contributed by the k-th route R_k and computed by the follow function.

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{S(R_k)}{dist(R_k)\cdot S_{avg}^{+}(G)} & \text{if route } R_k \text{ travels on edge } e_{i,j} \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

where $S(R_k)$ is the score of route R_k and $S^+_{avg}(G)$ is the maximum average score (computed by Eq. (5.4)) over the whole road network.

DPri-Sample 3: Instead of updating $p_{i,j}$ once a feasible route is sampled, we update $p_{i,j}$ when a set of m' routes have been sampled. Thus, $p_{i,j}$ is computed as below.

$$p_{i,j} = \frac{\eta_{i,j} \cdot \tau_{i,j}}{\Sigma_{v_h \in adj'_u(v_i)}(\eta_{i,h} \cdot \tau_{i,h})} \tag{5.8}$$

where the increase ratio $\tau_{i,j}$ is adjusted by $\tau'_{i,j}=(1-\rho)\cdot\tau_{i,j}+\Sigma^{m'}_{k=1}\Delta\tau^k_{i,j}$ and $\rho \in [0,1]$). Initially, we have $\tau_{i,j}=1$ for all edges $e_{i,j}$.

5.3.4 Sampling Quality Analysis

Assuming that the maximum degree of vertex in road network G is D, in the worst case, the total number of feasible routes is $|\mathcal{R}_L(s,d)| = D^{\frac{L}{L_{min}}}$, where L_{min} is the length of the shortest edge in G. For simplicity, we use ζ to denote $D^{\frac{L}{L_{min}}}$. In uniform sampling, each feasible route has the equal probability to be sampled. Therefore, in each iteration, the probability that the optimal route is selected is $Pr\{R^*\} = \frac{1}{\zeta}$. After I iterations, the probability that we can obtain the optimal route is $1-(1-\frac{1}{\zeta})^I$. To guarantee that the probability of obtaining the optimal result is larger than $(1-\varepsilon)$, $\varepsilon \in (0,1)$, the number of iterations should be $I \geqslant \frac{log\varepsilon}{log(\zeta-1)-log\zeta}$. The analysis of other sampling methods are similar, thus being omitted.

5.3.5 Complexity Analysis

Sampling a feasible route is the dominated time cost of the route sampling framework. Therefore, we focus on analysing the time cost for sampling feasible routes which utilize Eq. (5.2) to check the feasibility of an edge. The corresponding time complexity of this check is $O(\log n)$ because both $dist(R)$ and $|e_{i,j}|$ are already known, and $dist(SR_{j,d})$ can be retrieved with time $O(\log n)$. Therefore, the time complexity

for sampling a feasible route is $O(\frac{L}{L_{min}} \cdot D \cdot \log n)$, where $\frac{L}{L_{min}}$ is the maximum number of edges in the route, and D is the maximum vertex degree among all vertices.

5.4 Experiments

5.4.1 Setups

We conduct experiments over three real datasets, i.e., the road networks and POIs of Dublin, Beijing and London. All the datasets are extracted from OpenStreetMap and their details are presented in Table 5.2.

Table 5.2: Datasets for evaluating KCR query

Datasets	**No. of vertices**	**No. of edges**	**No. of POIs**
Dublin (DL)	62,975	82,730	5,297
Beijing (BJ)	46,029	62,778	21,192
London (LN)	209,045	282,267	34,341

In the experiments, we evaluate the performance of three route sampling strategies, i.e., uniform route sampling (Uni), prioritized route sampling (Pri), and dynamic prioritized route sampling (DPri). We use DPri1, DPri2 and DPri3 to denote the three different update strategies in DPri. In addition, we revise the Nearest Neighbor Heuristic (NNH) algorithm in [27] to process KCR query and use it as a baseline. NNH expands from the start location s, and selects the nearest POI containing query keywords to expand until reaching the distance threshold L.

We generate 50 KCR queries for each dataset. The two query locations of each query are randomly selected from vertices V, and query keywords are randomly selected from the vocabulary built based on the textual description of POIs. The average score of returned routes for 50 KCR queries is used as the metric to evaluate the performance of solutions.

The settings of used parameters are summarized in Table 5.3, where the default values are highlighted in bold font. We use London dataset as the default dataset if no specific statement. $|V(s,d)|$ denotes the number of vertices whose total distance

to two query locations, i.e., s and d, is less than the distance threshold L. $|V(s,d)|$ is computed with two expansions from s and d, respectively. In addition, the distance threshold of each KCR query is specified with a certain deviation from shortest distance $dist(SR_{s,d})$ between two query locations.

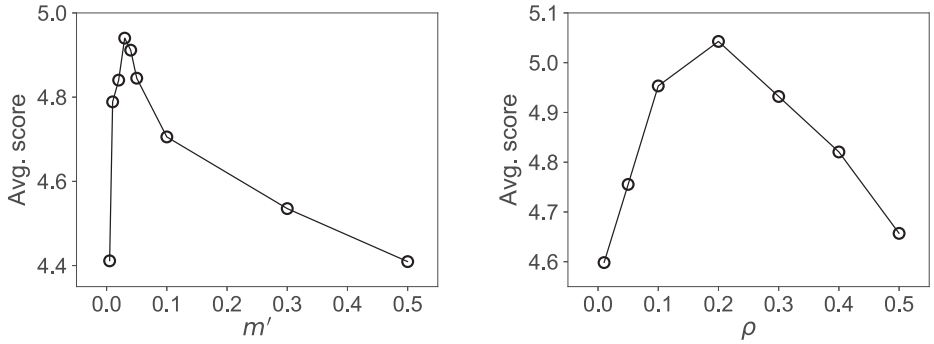
Table 5.3: Parameter setting in KCR query

Parameters	**Meaning**	**Values**
m'	number of sampled routes to update $p_{i,j}$	{0.005, 0.01, 0.02, **0.03**, 0.05, 0.1, 0.3, 0.5}$*$ $\|V(s,d)\|$
ρ	the decrease ratio in DPri3	0.01, 0.05, 0.1 **0.2**, 0.3, 0.4, 0.5
L	distance threshold	{1.2, **1.3**, 1.4, 1.5, 1.6}$*$ $dist(SR_{s,d})$
T	response time limit	**1**, 3, 5, 7, 9 (seconds)

5.4.2 Experimental Results

Varying parameters in DPri3: Figure 5.4 illustrates the average scores of 50 KCR queries over London dataset with T=1 second while varying the parameters m' and ρ. As suggested by Figure 5.4(a), the optimal value of m' is $0.03*|V(s,d)|$. A smaller m' will degrade DPri3 to the Pri sampling while a larger m' will slow the update of $p_{i,j}$, thus reducing the efficiency of route sampling. The other parameter used in DPri3 is the decrease ratio ρ which should be set to 0.2 according to Figure 5.4(b). A smaller ρ than 0.2 results in a quick increase of $p_{i,j}$ for frequently visited edges while reducing the chance to explore unvisited edges. In contrast, a larger ρ than 0.2 will weaken the utilization of previously sampled routes.

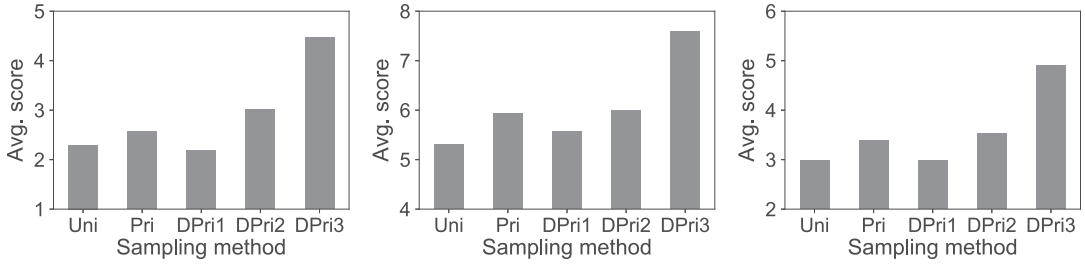
Comparing different route sampling methods: Figure 5.5 illustrates the performance of different route sampling methods on three datasets. Pri outperforms Uni because it considers the scores of edges rather than selecting edges uniformly. DPri1 does not achieve better performance than Pri because it has a low chance to visit the edges that have been visited many times before. However, these edges may still lead to the optimal routes. Differently, both DPri2 and DPri3 achieve better performance than other route sampling methods. Particularly, the query results returned



(a) Varying m' (b) Varying ρ

Figure 5.4: Varying parameters in DPri3.

by DPri3 are much better than that of other route sampling methods. Therefore, to simplify the presentation, we only report the results of DPri3 for dynamic prioritized route sampling in the subsequent experiments.

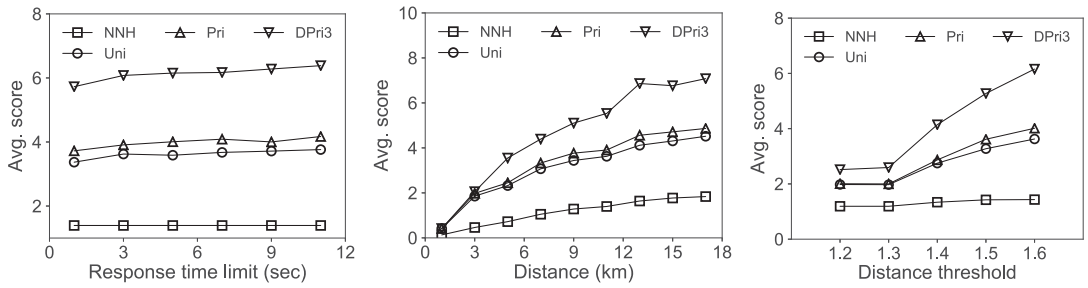


(a) Dublin (b) Beijing (c) London

Figure 5.5: Comparing different route sampling methods on three different datasets.

Varying response time: Figure 5.6(a) illustrates the results of KCR query while varying the response time limit. For Uni, Pri, and DPri3, the average route scores of query results gradually increases with the increase of response time limit. Actually, given a response time limit large enough, the route sampling framework can find results with very high quality. The query results obtained by NNH keep the same when varying the response time limit because it only generates one route greedily as the query result.

Varying distance $dist(SR_{s,d})$ and distance threshold L: We generate 20 distance intervals between 0 km and 20 km with a distance span of 1 km. For each distance interval, we randomly generate 50 KCR queries such that $dist(SR_{s,d})$ is



(a) Response time limit (b) $dist(SR_{s,d})$ (c) L

Figure 5.6: The results while varying response time limit, distance $dist(SR_{s,d})$ and distance threshold L

within the corresponding distance interval. According to Figure 5.6(b), the average score increases with the increase of $dist(SR_{s,d})$ and DPri3 outperforms Uni, Pri and NNH. Figure 5.6(c) illustrates the query results by varying the distance threshold L. As suggested by Figure 5.6(c), DPri3 still has the best performance.

5.5 Summary

To retrieve interesting routes for users, we propose KCR query that combines spatial keyword query and route planning to retrieve the routes that have the most query keywords while satisfying a cost constraint. To efficiently solve KCR query problem, we propose an adaptive route sampling framework with both static and dynamic route sampling techniques, where dynamic route sampling techniques can improve route sampling by learning knowledge from the routes sampled previously. The proposed route sampling framework can generate query results flexibly according to the given response time limit, thus avoiding the low efficiency of traditional exact solutions and the low quality of approximate solutions and achieving a tradeoff between the query efficiency and the quality of query results. Though KCR query in this work focuses on road networks, it can also find applications in many other scenarios, e.g., route searching in shopping malls and theme parks.

Chapter 6

Bounded-Cost Informative Route Query

6.1 Introduction

Although there exist some studies on route query over road networks using the spatio-textual data [8, 95, 54, 97, 53], they mainly aim to find a route to cover a set of query keywords while satisfying a travel cost (e.g., travel distance) budget or minimizing the travel cost. These queries are analogous to the Boolean keyword search in Web search engines and suitable for trip planning that aims at visiting particular types of POIs, e.g., a route passing a restaurant and a bank. However, if users want to find a route textually relevant to the given query keywords, keyword coverage search cannot apply because it only needs to cover the query keywords and cannot compute a textual relevance score for each route. Thus, existing studies cannot find a route that is most relevant to the given query keywords regarding the textual relevance.

In this Chapter, we propose a new route query termed *Bounded-Cost Informative Route (BCIR) query* which retrieves the optimal route that is most textually relevant to the user-specified query keywords. Concretely, given the start and destination locations s and d, the query keywords and a travel cost budget (e.g., the maximum travel distance or time), BCIR query finds the optimal route R^* from s to d such

that the travel cost of R^* is bounded by the given travel cost budget, and the textual relevance (also called route score) between the keyword description of R^* and the query keywords is maximized. The reason for introducing a travel cost constraint is that users generally have a cost budget in mind to avoid a high travel cost.

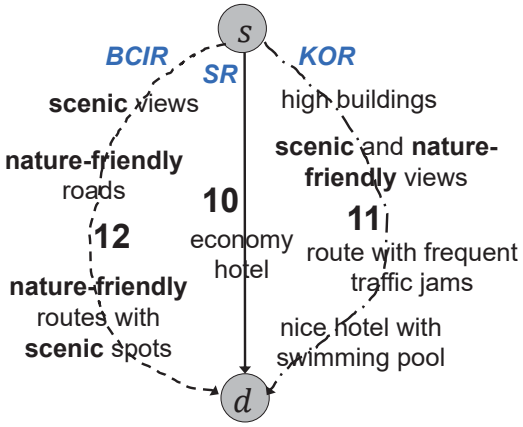


Figure 6.1: The difference between shortest route (SR) query, keyword-aware optimal route (KOR) query and bounded-cost informative route (BCIR) query, where query keywords are "*scenic, nature-friendly*".

For example, a tourist may issue a query with query keyword "*scenic, nature-friendly*" to find a route that is scenic and nature-friendly. Figure 6.1 illustrates the difference between shortest route (SR) query, keyword-aware optimal route (KOR) query [8] and BCIR query, where KOR query is one representative for the keyword coverage route queries. The number beside each route is the corresponding travel cost. The SR route is the shortest route from s to d but contains no query keywords. KOR query retrieves the optimal route that only needs to cover all the query keywords once. In contrast, the BCIR route is most relevant to "*scenic, nature-friendly*" than the other two queries with a little larger travel cost.

BCIR query has a wide range of potential applications, e.g., finding interesting tourist routes, finding emotional routes, and detecting the routes that are relevant to complaints to support urban maintenance. Particularly, two representative applications are discussed below.

Application scenario I: finding interesting tourist routes

Tourists usually would like to explore interesting attractions when visiting a new city. With the BCIR query, they can easily specify what they are interested in by query keywords and search for the corresponding route that is most relevant to the given query keywords. As discussed in Figure 6.1, one tourist may issue a BCIR query to find a route relevant to "*scenic, nature-friendly*" from her current location to the hotel. More interestingly, BCIR query can retrieve a route of some specific topics. For example, a route with the topic "*shopping*" can be retrieved by specifying some query keywords relevant to shopping such as "*sale*", "*mall*" and "*shop*".

Application scenario II: finding emotional routes

In addition to finding efficient routes that are short and fast, users may also want to find some routes that are emotionally pleasant [69]. Existing studies mainly focus on computing the emotion scores of routes from different aspects such as *happiness*, *quietness* and *beauty*. Novelly, BCIR offers a general and flexible way to compute an emotionally pleasant route by specifying some emotion-aware query keywords. For example, if someone wants to find a happy route, a BCIR query with query keywords "happy", "joy" and "delight" will help. Moreover, some dangerous and negative routes can also be identified by using query keywords like "*crime*", "*robbery*" and "*accident*" in BCIR query.

Challenges and Contributions

It is difficult to answer BCIR query efficiently due to its non-additive property (cf. Section 6.2.2), i.e., the textual relevance of a route is computed based on the text description of the entire route rather than simply summing the textual relevance values of its edges. This characteristic differentiates BCIR query from existing keyword coverage route queries which have additive route scores. Furthermore, BCIR query is actually an NP-hard problem (cf. Section 6.2.2). For such a computationally

expensive problem, we propose three solutions, i.e., an exact solution with efficient pruning techniques, an approximate solution with response time guarantee and another approximate solution with quality guarantee. The exact solution with multiple pruning methods can greatly reduce the search space and return exact results for BCIR queries of small cost budgets efficiently. In contrast, the two approximate solutions report good approximate results for BCIR queries of large cost budgets.

6.2 Problem Statement

In this section, we first formally define the BCIR query problem. Then, we prove that BCIR query is an NP-hard problem and discuss its non-additive property.

6.2.1 Problem Definition

Before defining BCIR query, we first give a new definition of road network as below.

Definition 6.1 (Road Network)**.** *A road network is modeled by a directed graph $G(V, E)$, where each vertex $v_i \in V$ represents an intersection of roads; each edge $e_{i,j} \in E$ represents the directed road segment from v_i to v_j and is associated with a travel cost $c_{i,j}$ and a set of keywords $\mathcal{K}_{i,j}$ with their occurrences.*

Example 6.1. *Figure 6.2 illustrates a small road network. For simplicity, we assume that all edges are bi-directed. The travel cost and keywords are labelled beside each edge. For example, the travel cost of edge $e_{s,3} = (s, v_3)$ is $c_{s,3}=5$ and its keywords are $\mathcal{K}_{s,3}=\{k_1$: 1, k_3: 1$\}$ where the numbers specify the occurrences of the corresponding keywords on edge $e_{s,3}$.*

The travel cost $c_{i,j}$ of edge $e_{i,j}$ can be any non-negative metric, such as the length of $e_{i,j}$ and the time to travel through $e_{i,j}$. The keywords $\mathcal{K}_{i,j}$ of edge $e_{i,j}$ can be extracted from different sources, e.g., the geo-tagged comments from Foursquare and Facebook.

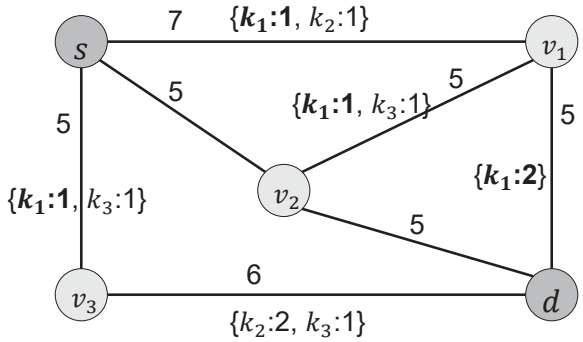


Figure 6.2: A small road network, where each edge is associated with its travel cost and keywords (if exist). Given a BCIR query with start location s, destination d, query keyword $\mathcal{K}_q$=$\{k_1\}$, and travel cost budget B=12, the query result is route R^*=$\langle s, v_1, d\rangle$.

A route R=$\langle v_{x_1}, \ldots, v_{x_\gamma}\rangle$ consists of a sequence of vertices, where edge $(v_{x_i}, v_{x_{i+1}})$∈E and v_{x_i}≠v_{x_j} if x_i≠x_j $(1 \leqslant i, j \leqslant \gamma)$. We use E_R and V_R to denote those edges and vertices on route R, respectively. The travel cost of R, $\lambda(R)$, is computed by summing the travel costs of its edges, i.e.,

$$\lambda(R) = \sum_{e_{i,j} \in E_R} c_{i,j}. \tag{6.1}$$

The shortest route from v_i to v_j is denoted by $SR_{i,j}$ whose travel cost is $\lambda(SR_{i,j})$. The keyword description of route R, $\mathcal{K}_R$, is computed by merging the keywords of its edges, i.e.,

$$\mathcal{K}_R = \biguplus_{e_{i,j} \in E_R} \mathcal{K}_{i,j}, \tag{6.2}$$

where $\biguplus$ is an union operation for multi-set that allows duplicate elements.

The BCIR query is then defined as below.

Definition 6.2 (Bounded-Cost Informative Route (BCIR) Query)**.** *Given a road network G, a BCIR query is denoted by q=$(s, d, \mathcal{K}_q, B)$, where s and d specify the start location and destination, respectively, $\mathcal{K}_q$ is a set of query keywords, and B is*

a travel cost budget. The objective of q is to find the optimal route R^* from s to d such that:

$$R^* = arg \max_{R \in \mathcal{C}^B_{s,d}} \tau(R) \tag{6.3}$$

where $\tau(R)$ computes the textual relevance between $\mathcal{K}_R$ and $\mathcal{K}_q$; $\mathcal{C}^B_{s,d}$ represents those candidate routes, from s to d, whose travel costs are less than or equal to B, i.e.,

$$\mathcal{C}^B_{s,d} = \{R | R \in \mathcal{R}_{s,d} \wedge \lambda(R) \leqslant B\} \tag{6.4}$$

and $\mathcal{R}_{s,d}$ represents the entire set of simple routes from s to d.

The cost budget B is specified by a deviation ratio $\mu \geqslant 0$ from the shortest route between two query locations, i.e., $B{=}(1 + \mu) \cdot \lambda(SR_{s,d})$. Note that the travel cost budget B can be also specified by users directly.

In this work, we compute route score $\tau(R)$ by utilizing the TF-IDF model [104] which is widely used for information retrieval. Concretely, we have

$$\tau(R) = \frac{\sum_{k \in \mathcal{K}_q} (w_{k,R}) \cdot (w_{k,q})}{\sqrt{\sum_{k \in \mathcal{K}_R} (w_{k,R})^2 \cdot \sum_{k \in \mathcal{K}_q} (w_{k,q})^2}} \tag{6.5}$$

where $w_{k,R}{=}1 + \ln(f_{k,R})$ and $f_{k,R}$ counts the occurrences of keyword k in $\mathcal{K}_R$; $w_{k,q}{=}\ln(1 + \frac{|E|}{|E_k|})$ and E_k are those edges containing keyword k.

Example 6.2. *Figure 6.2 illustrates a BCIR query, where the start location and destination are s and d, respectively; $\mathcal{K}_q{=}\{k_1\}$ specifies one query keyword, and the travel cost budget is $B{=}12$. As listed in Table 6.1, there are five possible routes. Particularly, for route $R_1{=}\langle s, v_1, d\rangle$, we have*

$$\mathcal{K}_{R_1} = \mathcal{K}_{s,1} \uplus \mathcal{K}_{1,d} = \{\langle k_1 : 3\rangle, \langle k_2 : 1\rangle\}$$

where the numbers record the occurrences of the corresponding keywords. For example, $\langle k_1 : 3\rangle$ indicates that keyword k_1 appears three times on route R_1. We then

have $w_{k_1,R_1}=1+\ln 3$ *and* $w_{k_2,R_1}=1+\ln 1$. *Meanwhile, we have* $|E_{k_1}|=4$, $|E_{k_2}|=2$ *and* $|E_{k_3}|=3$, *which count the numbers of edges containing keywords* k_1, k_2 *and* k_3, *respectively. For query keyword* k_1, *we have*

$$w_{k_1,q} = \ln(1+\frac{|E|}{|E_{k_1}|}) = \ln(1+\frac{7}{4}) = \ln\frac{11}{4}$$

Accordingly, the score of route R_1 *is*

$$\begin{aligned}\tau(R_1) &= \frac{\sum_{k\in\mathcal{K}_q}(w_{k,R_1})\cdot(w_{k,q})}{\sqrt{\sum_{k\in\mathcal{K}_{R_1}}(w_{k,R_1})^2\cdot\sum_{k\in\mathcal{K}_q}(w_{k,q})^2}}\\ &= \frac{(1+\ln 3)\cdot(\ln\frac{11}{4})}{\sqrt{((1+\ln 3)^2+(1+\ln 1)^2)\cdot(\ln\frac{11}{4})^2}}\\ &= 0.903\end{aligned}$$

The scores of the other four routes can be computed in the same way. The details of five routes are summarized in Table 6.1. Routes R_4 *and* R_5 *should be pruned since their travel costs are larger than the cost budget* B=12. *The other three routes are candidate routes and* R_1 *is returned as the query result because its score is larger than that of routes* R_2 *and* R_3.

Table 6.1: All possible routes for the BCIR query in Figure 1

Route	**Cost**	**Keyword**	**Score**
$R_1=\langle s,v_1,d\rangle$	12	$\{\langle k_1:3\rangle,\langle k_2:1\rangle\}$	0.903
$R_2=\langle s,v_2,d\rangle$	10	$\{\}$	0
$R_3=\langle s,v_3,d\rangle$	11	$\{\langle k_1:1\rangle,\langle k_2:2\rangle,\langle k_3:2\rangle\}$	0.385
$R_4=\langle s,v_2,v_1,d\rangle$	15	$\{\langle k_1:3\rangle,\langle k_3:1\rangle\}$	0.903
$R_5=\langle s,v_1,v_2,d\rangle$	17	$\{\langle k_1:2\rangle,\langle k_2:1\rangle,\langle k_3:1\rangle\}$	0.767

6.2.2 Problem Hardness

A. NP-hardness

BCIR query can be proved to be NP-hard. First, we define the decision problem of BCIR query, Decision-BCIR, as below.

Definition 6.3 (Decision-BCIR)**.** *Given a road network G, a BCIR query q=$(s, d, \mathcal{K}_q, B)$ and a score threshold X>0, Decision-BCIR decides whether there exists a route from s to d such that its cost is at most B and its score is at least X.*

Then, we have the following theorem.

Theorem 6.1. *Decision-BCIR problem is NP-hard.*

Proof. This theorem can be proved by a reduction from the Hamiltonian Route/Path (Ham-Route for short) problem, a well-known NP-Complete problem, to the Decision-BCIR problem. Given a road network $G(V, E)$, Ham-Route problem decides whether there exists a route passing each vertex in V once and only once. Then, we can formulate an instance of the Decision-BCIR problem based G as below.

- **Road network** $G'(V', E')$**:** We create a new road network $G'(V', E')$ by letting V'=$V \cup \{v_s, v_d\}$ and E'=$E \cup \{(v_i, v_j)|v_i \in \{v_s, v_d\} \wedge v_j \in V\}$, where v_s and v_d are two new vertices. For each edge $e_{i,j} \in E'$, we set its cost $c_{i,j}$=1. In addition, we assume that edge $e_{i,j} \in E$ in G' contains one unique keyword and edge $e_{i,j} \in \{(v_i, v_j)|v_i \in \{v_s, v_d\} \wedge v_j \in V\}$ does not contain any keywords.

- **Decision-BCIR problem** q=$(s, d, \mathcal{K}_q, B, X)$**:** The start location s and destination d correspond to vertices v_s and v_d on G', respectively. Query keywords $\mathcal{K}_q$ contains all the keywords in G', i.e., $|\mathcal{K}_q|$=$|E|$. The cost budget B and score threshold X are set to $n + 1$ and $\sqrt{\frac{n-1}{|E|}}$, respectively, where n=$|V|$.

Then, we can prove that G has a Ham-Route R if and only if there exists a route R' from v_s to v_d on G' such that the cost of R' is at most $n + 1$ and the score of R' is at least $\sqrt{\frac{n-1}{|E|}}$.

One the one hand, assuming that there is a Ham-Route R in G that passes all the n vertices in V, we can get a new route R' in G' by adding v_s and v_d to the two

ends of R, respectively. The cost of R' is $n+1$ because R' has $n+2$ vertices and the cost of each edge is exactly 1. Meanwhile, the score of R' is

$$\tau(R') = \frac{\sum_{k\in(\mathcal{K}_{R'}\cap\mathcal{K}_q)}(w_{k,R'})\cdot(w_{k,q})}{\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2\cdot\sum_{k\in\mathcal{K}_q}(w_{k,q})^2}}. \tag{6.6}$$

Considering that each edge (except for the edges connecting to v_s and v_d) in G' contains one unique keyword, we have $|E'_k|$=1 for keyword k and $|\mathcal{K}_{R'}|$=$n-1$. Furthermore, we have $w_{k,q}$=$\ln(1+\frac{|E'|}{|E'_k|})$=$\ln(1+|E'|)$ and $w_{k,R'}$=$1+\ln(f_{k,R'})$=$1+\ln(1)$=1. We then have

$$\begin{aligned}\tau(R') &= \frac{\sum_{k\in(\mathcal{K}_{R'}\cap\mathcal{K}_q)}1\cdot\ln(1+|E'|)}{\sqrt{\sum_{k\in\mathcal{K}_{R'}}1^2\cdot\sum_{k\in\mathcal{K}_q}(\ln(1+|E'|))^2}}\\ &= \frac{|\mathcal{K}_{R'}|\cdot\ln(1+|E'|)}{\sqrt{|\mathcal{K}_{R'}|\cdot|\mathcal{K}_q|}\cdot\ln(1+|E'|)}\\ &= \sqrt{\frac{|\mathcal{K}_{R'}|}{|\mathcal{K}_q|}} = \sqrt{\frac{n-1}{|E|}}\end{aligned} \tag{6.7}$$

where $\sum_{k\in(\mathcal{K}_{R'}\cap\mathcal{K}_q)}(w_{k,R'})\cdot(w_{k,q}) = |\mathcal{K}_{R'}|\cdot\ln(1+|E'|)$ because $\mathcal{K}_q$ contains all the keywords and $\mathcal{K}_{R'}\subseteq\mathcal{K}_q$. Therefore, if there is a Ham-Route R in G, we can find a route R' in G' such that its cost is at most $n+1$ and its score is at least $\sqrt{\frac{n-1}{|E|}}$.

On the other hand, we assume that there is a route R' in G' such that its cost is at most $n+1$ and its score is at least $\sqrt{\frac{n-1}{|E|}}$. According to Eq. (6.7), we have $\tau(R')$=$\sqrt{\frac{|\mathcal{K}_{R'}|}{|\mathcal{K}_q|}}$=$\sqrt{\frac{|\mathcal{K}_{R'}|}{|E|}}$, where $|\mathcal{K}_{R'}|$ counts the number of keywords on route R'. R' should contain at least $n-1$ keywords if its score is at least $\sqrt{\frac{n-1}{|E|}}$. Accordingly, R' must have at least $n+1$ edges considering that each edge in G' contains one unique keyword and the edges connected to v_s and v_d have no keywords. Meanwhile, R' has at most $n+1$ edges because its cost is at most $n+1$ and the cost of each edge is 1.

Therefore, R' has exactly $n+1$ edges. By removing v_s and v_d from R', we get the corresponding Ham-Route R in G.

It completes the proof. □

Therefore, Desicion-BCIR problem is NP-hard, thus the NP-hardness of BCIR query.

B. Non-additive Property

The route score in BCIR query is non-additive because it is computed based on the text description of the entire route. In other words, the score of a route cannot be computed by adding the scores of its sub-routes.

Example 6.3. *Take the BCIR query in Figure 6.2 for example and consider route* R_1=$\langle s, v_1, d\rangle$ *and its sub-routes* R_{11}=$\langle s, v_1\rangle$ *and* R_{12}=$\langle v_1, d\rangle$. *With Eq. (6.5), we have* $\tau(R_1)$=0.903, $\tau(R_{11})$=0.707, *and* $\tau(R_{12})$=1.0. *Obviously, the score of route* R_1 *cannot be computed by summing the scores of its sub-routes, i.e.,* $\tau(R_1)\neq\tau(R_{11}) + \tau(R_{12})$.

The non-additive property makes BCIR query more difficult than the existing keyword coverage route queries [8, 53, 54, 95, 97] in which the scores of routes are additive. Accordingly, this non-additive property also prevents us from leveraging existing techniques to process BCIR queries efficiently.

6.3 Solution Overview

Considering the hardness of BCIR query problem, we design different solutions for different application scenarios.

Scenario I: Requiring optimal result

In this scenario, users request for the exact results. BCIR query is NP-hard, indicating that it is impossible to devise an exact solution of polynominal complexity to

process BCIR query. Nevertheless, it is still possible to evaluate all candidate routes if the travel cost budget is small. Additionally, some users may be willing to wait for a reasonably long time for the exact query results. Therefore, we propose an exact solution to BCIR query problem and design effective pruning techniques to reduce the search space

Scenario II: Limiting response time

In this scenario, users wish to obtain the query results within a specified time limit. To satisfy this requirement, we propose the time-bounded solution (TBS) which receives as input a processing time limit and aims at maximizing the scores of returned routes within the time limit.

Scenario III: Guaranteeing answer quality

Some users would like to sacrifice a certain degree of exactness to reduce the running time as long as the quality of query result is guaranteed. To meet this requirement, we propose the error-bounded solution (EBS) which imposes an approximation error threshold on the returned route.

Figure 6.3 provides an overview of the three solutions. Sections 6.4, 6.5 and 6.6 will discuss the technical details for the exact solution, time-bounded solution and error-bounded solution, respectively.

6.4 Exact Solution

6.4.1 Algorithm Sketch for Exact Solution

BCIR query retrieves the optimal route with the largest score among all the candidate routes $\mathcal{C}_{s,d}^{B}$ that satisfy the travel cost budget B. Therefore, one exact solution for solving BCIR query problem is to evaluate all the candidate routes and select the route with the largest score as the query result. Algorithm 6.1 sketches this exact solution which explores the candidate routes by a depth-first expansion from the

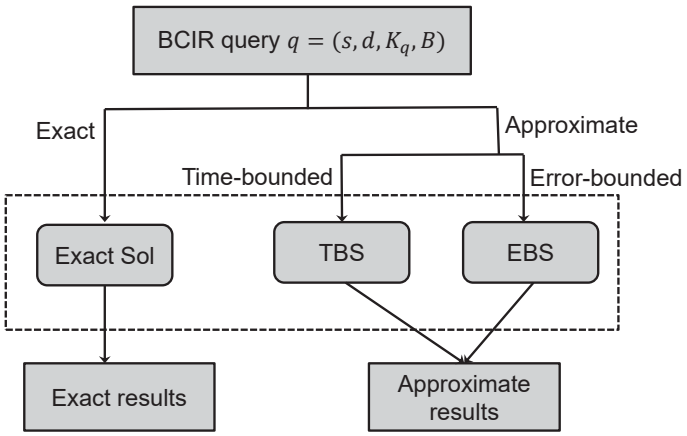


Figure 6.3: Solution overview for BCIR query

start location s. Initially, a partial route set U is created to store partial routes (i.e., routes starting from s but not reaching destination d) during the query processing and R^* is used to record the current optimal route. The initialization of U and R^* will be discussed in Section 6.4.5. In each iteration, a partial route is selected from U to generate more partial routes by expanding the adjacent vertices of its end vertex. Once a partial route reaches the destination d and its score is larger than the current optimal route R^*, R^* is updated. The algorithm terminates until U is empty.

However, it is computationally expensive to conduct such an exploration because the number of candidate routes could be considerably large. To reduce the search space, we design effective pruning techniques to avoid checking those candidate routes that cannot be the optimal result as soon as possible (line 6 in Algorithm 6.1, cf. Sections 6.4.3 and 6.4.4).

6.4.2 Indexing

In order to facilitate the query processing, we build a *Shortest Route Index* and an *Inverted Index* for road network G.

Shortest route index: Shortest route computation will be frequently used to check whether a route is feasible when processing the BCIR query. In order to facilitate

Algorithm 6.1 Algorithm sketch for the exact solution

Input: BCIR query q=$(s, d, \mathcal{K}_q, B)$
Output: The optimal route R^*

```
1: Initialize partial route set U
2: Initialize the optimal route R* by heuristics                ▷ Section 6.4.5
3: while U is not empty do
4:     select a partial route R from U
5:     generate new partial routes C_R based on R
6:     prune unpromising routes in C_R                          ▷ Sections 6.4.3 and 6.4.4
7:     for R' ∈ C_R do
8:         if R' reaches the destination d then
9:             update R*
10:        else
11:            add R' to U
12: return R*
```

the shortest route computation, we build a Contraction Hierarchy (CH) [36] index, one of the most efficient index structures for shortest route computation, for road network G.

Inverted index: In order to check which edges contain the query keywords, an inverted index is built for all edges E. Inverted index is widely used for text indexing by listing for each keyword those documents that contain it. Table 6.2 is an example of the inverted index for the road network in Figure 6.2, where the numbers after the colon are the occurrences of the corresponding keywords.

Table 6.2: The inverted index for the edges in Figure 6.2

Keywords	**Edge list**
k_1	$\langle e_{s,1} : 1\rangle$, $\langle e_{s,3} : 1\rangle$, $\langle e_{1,2} : 1\rangle$, $\langle e_{1,d} : 2\rangle$
k_2	$\langle e_{s,1} : 1\rangle$, $\langle e_{3,d} : 2\rangle$
k_3	$\langle e_{s,3} : 1\rangle$, $\langle e_{1,2} : 1\rangle$, $\langle e_{3,d} : 1\rangle$

Caching shortest costs: In addition, during the query processing of each BCIR query, we build a hash table S to cache the shortest costs, $\lambda(SR_{i,j})$, that have been computed. Then, $\lambda(SR_{i,j})$ can be obtained from S directly if it is in the hash table, which will further facilitate the query processing. The corresponding algorithm for looking up $\lambda(SR_{i,j})$ is detailed in Algorithm 6.2.

In what follows, we will detail the proposed pruning techniques (Sections 6.4.3

Algorithm 6.2 Look up the shortest cost $\lambda(SR_{i,j})$

Input: Cached hash table S, start vertex v_i and end vertex v_j
Output: The shortest cost $\lambda(SR_{i,j})$

```
1: λ(SR_{i,j}) ← ∞
2: if S contains λ(SR_{i,j}) then
3:     get λ(SR_{i,j}) from S
4: else
5:     compute SR_{i,j} by CH index
6:     for each vertex v_x on route SR_{i,j} do
7:         if S does not contains λ(SR_{i,x}) then
8:             add λ(SR_{i,x}) to S
9: return λ(SR_{i,j})
```

and 6.4.4) and the corresponding query processing algorithm that combines these techniques (Section 6.4.5).

6.4.3 Cost Pruning

The goal of cost pruning is to prune those partial routes that violate the cost budget B during query processing. In other worlds, the pruned partial routes cannot expand to generate any candidate routes.

Given a partial route $R{=}\langle s, \ldots, v_i\rangle$, the set of candidate routes, from the start location s to the destination d, generated by expanding R is

$$\mathcal{C}_R = \{R' | R' = R \oplus R_x, R_x \in \mathcal{C}_{i,d}^{B'}\} \tag{6.8}$$

where $C_{i,d}^{B'}$ is the set of candidate routes from the end vertex v_i to the destination d (cf. Eq. (6.4)), $B'{=}B - \lambda(R)$, and $\oplus$ is an operation to concatenate two routes.

The essential idea of cost pruning is to compute a lower bound cost $\lambda^-(R)$ for all the candidate routes, $\mathcal{C}_R$, generated by expanding partial route R, and verify whether $\lambda^-(R)$ violates the cost budget B. First, we define the lower bound cost $\lambda^-(R)$.

Definition 6.4 (Lower bound cost)**.** *Given partial route $R{=}\langle s, \ldots, v_i\rangle$, a lower bound cost $\lambda^-(R)$ for all the candidate routes $\mathcal{C}_R$ satisfies $\lambda^-(R) \leqslant \lambda(R')$, $\forall R' \in \mathcal{C}_R$.*

With the lower bound cost $\lambda^-(R)$, the partial route R should be pruned if $\lambda^-(R) > B$.

Considering that the shortest cost between two vertices is a lower bound for all the routes between them, we compute $\lambda^-(R)$ as below.

$$\lambda^-(R) = \lambda(R) + \lambda(SR_{i,d}) \tag{6.9}$$

where $SR_{i,d}$ is the shortest route from the end vertex v_i to the destination d. We then have the following lemma.

Lemma 6.1. *Given partial route R, the $\lambda^-(R)$ computed by Eq. (6.9) is a lower bound for the costs of those candidate routes $\mathcal{C}_R$ generated by expanding R.*

Proof. For any candidate route $R' \in \mathcal{C}_R$, we assume that $R' = R \oplus R_x$, where $R_x \in \mathcal{C}_{i,d}^{B'}$. Since $SR_{i,d}$ is the shortest route from v_i to d, we have $\lambda(SR_{i,d}) \leqslant \lambda(R_x), \forall R_x \in \mathcal{C}_{i,d}^{B'}$. Therefore, for any candidate route $R' \in \mathcal{C}_R$, we have $\lambda(R') = \lambda(R) + \lambda(R_x) \geqslant \lambda(R) + \lambda(SR_{i,d}) = \lambda^-(R)$. Thus, $\lambda^-(R)$ is a lower bound for the costs of those candidate routes generated by expanding partial route R. □

Example 6.4. *Take the BCIR query in Figure 6.2 for example. Assuming that the partial route is $R = \langle s, v_2, v_1 \rangle$, R should be pruned since $\lambda^-(R) = \lambda(R) + \lambda(SR_{1,d}) = 10 + 5 > B$, where the cost budget B=12.*

Pre-computing $\lambda(SR_{i,d})$: When computing $\lambda^-(R)$, $\lambda(SR_{i,d})$ is computed online, which incurs a high computation cost. With the observation that the destination d is fixed for all shortest routes $SR_{i,d}$, we pre-compute the smallest costs to the destination d for those vertices that are within the travel cost budget from d by utilizing a reverse Dijkstra'a algorithm. With these computed smallest costs, $\lambda(SR_{i,d})$ can be retrieved directly instead of being computed from the scratch during the query processing.

6.4.4 Score Pruning

Intuitively, a partial route R should be pruned if all of its expanded candidate routes, $\mathcal{C}_R$, are not better than the current optimal route R^*. Therefore, we aim at computing an upper bound score, $\tau^+(R)$, for all the candidate routes $\mathcal{C}_R$ and exploit $\tau^+(R)$ to verify the possibility that there exists a route in $\mathcal{C}_R$ that is better than the current optimal route R^*.

First, we define the upper bound score $\tau^+(R)$ as below.

Definition 6.5 (Upper bound score)**.** *Given partial route $R{=}\langle s, \ldots, v_i\rangle$, an upper bound score $\tau^+(R)$ for all the candidate routes in $\mathcal{C}_R$ satisfies $\tau^+(R){\geqslant}\tau(R')$, $\forall R' \in \mathcal{C}_R$.*

Then, we have the following lemma about score pruning.

Lemma 6.2 (Score Pruning)**.** *Given partial route R and the current optimal route R^*, R should be pruned if $\tau^+(R) \leqslant \tau(R^*)$, where $\tau^+(R)$ is an upper bound score for all the candidate routes generated by expanding R, i.e., $\mathcal{C}_R$.*

Proof. Since $\tau^+(R)$ is an upper bound score for all the candidate routes in $\mathcal{C}_R$, we have $\tau(R'){\leqslant}\tau^+(R)$ for each route $R'{\in}\mathcal{C}_R$. By inequality transition, we have $\tau(R'){\leqslant}\tau(R^*)$, indicating that the candidate routes generated by expanding partial route R cannot be better than the current optimal route R^*. Therefore, R should be pruned. □

Computing $\tau^+(R)$

The challenge for computing $\tau^+(R)$ is twofold. On the one hand, $\tau^+(R)$ should be computed based on a route set, $\mathcal{C}_R$, rather than a single route. On the other hand, as discussed in Section 6.2.2, the route score in BCIR query is non-additive and we cannot add the scores of routes R_x and R_y to obtain the score of route $R_x \oplus R_y$. With these two challenges, the proposed techniques for computing the upper bounds of route scores in existing studies cannot be applied to the BCIR query directly.

In this study, according to the definition of route score in Eq. (6.5), we compute the upper bound score $\tau^+(R)$ by the following equation.

$$\tau^+(R) = \frac{\sum_{k \in \mathcal{K}_q} (1 + \ln(f_{k,R} + F^+(k, i, d, B'))) \cdot (w_{k,q})}{\sqrt{\sum_{k \in \mathcal{K}_R} (1 + \ln(f_{k,R}))^2 \cdot \sum_{k \in \mathcal{K}_q} (w_{k,q})^2}} \tag{6.10}$$

where $f_{k,R}$ is the frequency of keyword k on route R, $F^+(k, i, d, B')$ is a function that computes an upper bound for the occurrences of keyword k on all the sub-routes $\mathcal{C}_{i,d}^{B'}$ and requires $f_{k,R_x} \leqslant F^+(k, i, d, B'), \forall R_x \in \mathcal{C}_{i,d}^{B'}$. We will discuss the computation of $F^+(k, i, d, B')$ later.

Then, we have the following lemma about the upper bound score computed above.

Lemma 6.3. *The upper bound score $\tau^+(R)$ computed by Eq. (6.10) is an upper bound for the scores of all the candidate routes, $\mathcal{C}_R$, generated by expanding partial route R, i.e., $\tau(R') \leqslant \tau^+(R), \forall R' \in \mathcal{C}_R$.*

Proof. We use R' to represent any candidate route in $\mathcal{C}_R$ and let $R' = R \oplus R_x$, where sub-route $R_x \in \mathcal{C}_{i,d}^{B'}$. Accordingly, the frequency of keyword k on route R' is the sum of the frequencies of keyword k on routes R and R_x, i.e., $f_{k,R'} = f_{k,R} + f_{k,R_x}$.

According to the definition of route score in Eq. (6.5), we have

$$\tau(R') = \frac{\sum_{k \in \mathcal{K}_q} (w_{k,R'}) \cdot (w_{k,q})}{\sqrt{\sum_{k \in \mathcal{K}_{R'}} (w_{k,R'})^2 \cdot \sum_{k \in \mathcal{K}_q} (w_{k,q})^2}} \tag{6.11}$$

For simplicity, we denote $w_q = \sqrt{\sum_{k \in \mathcal{K}_q} (w_{k,q})^2}$ and then have

$$\tau(R') = \frac{\sum_{k \in \mathcal{K}_q} (w_{k,R'}) \cdot (w_{k,q})}{w_q \cdot \sqrt{\sum_{k \in \mathcal{K}_{R'}} (w_{k,R'})^2}} \tag{6.12}$$

On the one hand, for Eq. (6.12), we have

$$\sum_{k \in \mathcal{K}_q} (w_{k,R'}) \cdot (w_{k,q}) = \sum_{k \in \mathcal{K}_q} (1 + \ln(f_{k,R} + f_{k,R_x})) \cdot (w_{k,q}) \tag{6.13}$$

Since $F^+(k,i,d,B')$ computes an upper bound for the occurrences of keyword k for all the sub-routes $R_x \in \mathcal{C}_{i,d}^{B'}$, we have

$$\sum_{k\in\mathcal{K}_q}(w_{k,R'})\cdot(w_{k,q}) \leqslant \sum_{k\in\mathcal{K}_q}(1+\ln(f_{k,R}+F^+(k,i,d,B')))\cdot(w_{k,q}) \tag{6.14}$$

On the other hand, the denominator in Eq. (6.12) has the following derivation.

$$\begin{aligned} w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2} &= w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_{R'}}(1+\ln(f_{k,R}+f_{k,R_x}))^2} \\ &\geqslant w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_R}(1+\ln(f_{k,R}))^2} \end{aligned} \tag{6.15}$$

Combining Eq. (6.12), Eq. (6.14) and Eq. (6.15), we have

$$\begin{aligned} \tau(R') &= \frac{\sum_{k\in\mathcal{K}_q}(w_{k,R'})\cdot(w_{k,q})}{w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2}} \\ &\leqslant \frac{\sum_{k\in\mathcal{K}_q}(1+\ln(f_{k,R}+F^+(k,i,d,B')))\cdot(w_{k,q})}{w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_R}(1+\ln(f_{k,R}))^2}} \\ &= \tau^+(R) \end{aligned}$$

Therefore, we have $\tau(R')\leqslant\tau^+(R)$ for any candidate route $R'\in\mathcal{C}_R$. Thus, the $\tau^+(R)$ computed by Eq. (6.10) is an upper bound for the scores of all the candidate routes, $\mathcal{C}_R$, generated by expanding partial route R. □

Computing $F^+(k,i,d,B')$

Given partial route $R=\langle s,\ldots,v_i\rangle$, function $F^+(k,i,d,B')$ computes an upper bound for the occurrences of keyword k on all the sub-routes $\mathcal{C}_{i,d}^{B'}$, where i and d are the subscript of the end vertex v_i of R and the destination d, and $B'=B-\lambda(R)$. Since $F^+(k,i,d,B')$ will be invoked frequently during query processing to compute the upper bound score, an efficient method for computing $F^+(k,i,d,B')$ is required.

One straightforward method for computing $F^+(k,i,d,B')$ is to evaluate the occurrences of keyword k, f_{k,R_x}, on each route $R_x \in \mathcal{C}_{i,d}^{B'}$ and report the maximum f_{k,R_x}. However, $\mathcal{C}_{i,d}^{B'}$ could be considerably large, making it time prohibitive to evaluate all the routes in $\mathcal{C}_{i,d}^{B'}$. Therefore, instead of evaluating all the routes in $\mathcal{C}_{i,d}^{B'}$, we utilize those edges on routes $\mathcal{C}_{i,d}^{B'}$ to directly estimate the maximum possible number of occurrences of keyword k on each route $R_x \in \mathcal{C}_{i,d}^{B'}$.

With end vertex v_i, destination d and the remaining cost budget B', the set of edges $E_{i,d}^{B'}$ that appear on routes $\mathcal{C}_{i,d}^{B'}$ and contain keyword k is computed by

$$E_{i,d}^{B'}(k) = \{e_{f,g} | e_{f,g} \notin R \wedge k \in \mathcal{K}_{f,g} \wedge \Gamma(i,f,g,d) \leqslant B'\} \tag{6.16}$$

where function $\Gamma(i,f,g,d) = \lambda(SR_{i,f}) + c_{f,g} + \lambda(SR_{g,d})$ computes the smallest travel cost from v_i to d when passing edge $e_{f,g}$.

In Eq. (6.16), inverted index is utilized to check whether edge $e_{f,g}$ contains query keyword k and Algorithm 6.2 is invoked to compute $\lambda(SR_{i,f})$. Note that we can get $\lambda(SR_{g,d})$ in Eq. (6.16) directly since a reverse Dijkstra'a algorithm has been conducted at the beginning of query processing to compute the shortest cost to the destination d for each vertex (cf. the last paragraph in Section 6.4.3).

Straightforwardly, $F^+(k,i,d,B')$ can be computed by counting the total occurrences of k on all edges in $E_{i,d}^{B'}(k)$. Nonetheless, the computed upper bound $F^+(k,i,d,B')$ in this way could be very loose since $E_{i,d}^{B'}(k)$ covers much more edges than each route $R_x \in \mathcal{C}_{i,d}^{B'}$.

Example 6.5. *Take Figure 6.4 for example and assume that the remaining cost budget is $B'=6$. With Eq. (6.16), the set of edges on routes $\mathcal{C}_{i,d}^{B'}$ and containing keyword k is $E_{i,d}^{B'}(k)=\{e_{1,2}, e_{2,3}, e_{5,7}, e_{7,8}, e_{8,d}\}$ whose total keyword occurrences are 20. However, the corresponding total cost is 9 which is larger than the remaining cost budget $B'=6$, indicating that the computed upper bound is loose.*

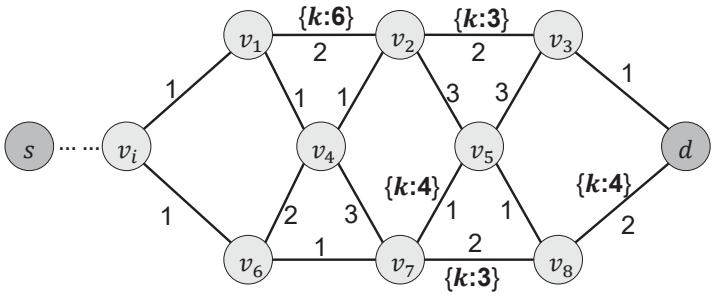


Figure 6.4: Road network example for computing $F^+(k, i, d, B')$.

To obtain a tighter $F^+(k, i, d, B')$, we compute the maximum number of keywords k that each route $R_x \in \mathcal{C}_{i,d}^{B'}$ could have by selecting a sub-set of edges from $E_{i,d}^{B'}(k)$ rather than using the whole set. To this end, we build an average keyword frequency histogram $\mathcal{H}$ for those edges in $E_{i,d}^{B'}(k)$ and leverage this histogram to compute $F^+(k, i, d, B')$.

First, with the frequency of keyword k on edge e, $f_{k,e}$, the corresponding average keyword frequency is $\bar{f}_{k,e} = \frac{f_{k,e}}{c_e}$, where c_e is the travel cost of edge e.

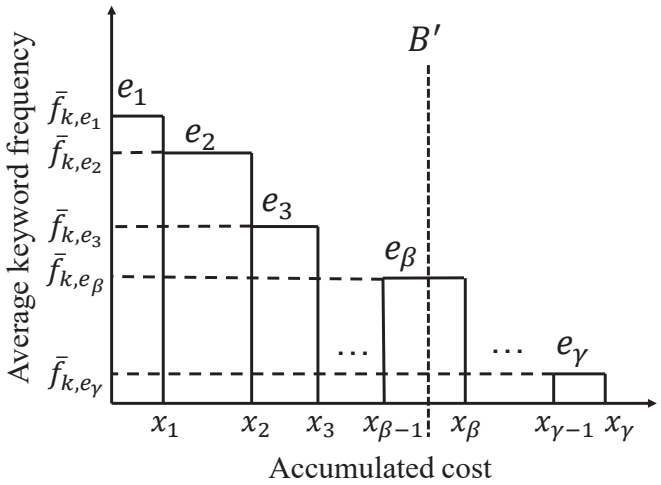


Figure 6.5: Average keyword frequency histogram

All edges in $E_{i,d}^{B'}(k)$ are then sorted based on their average keyword frequencies. Without loss of generality, we assume that the sorted edges are $e_1, e_2, \ldots, e_\gamma$ where $\bar{f}_{k,e_i} \leqslant \bar{f}_{k,e_j}$ if $i<j$. The corresponding average keyword frequency histogram $\mathcal{H}$ for these sorted edges is illustrated in Figure 6.5, where the accumulated cost $x_i = x_{i-1} +$

c_{e_i} and x_1=c_{e_1}.

Given the remaining travel cost budget B' and the average keyword frequency histogram $\mathcal{H}$, $F^+(k,i,d,B')$ is computed by

$$\begin{aligned} F^+(k,i,d,B') &= \bar{f}_{k,e_1} \cdot x_1 + \bar{f}_{k,e_2} \cdot (x_2 - x_1) + \cdots + \bar{f}_{k,e_\beta} \cdot (B' - x_{\beta-1}) \\ &= f_{k,e_1} + f_{k,e_2} + \cdots + \bar{f}_{k,e_\beta} \cdot (B' - x_{\beta-1}) \end{aligned} \tag{6.17}$$

where B' is assumed to be located at edge e_β, i.e., $x_{\beta-1} < B' \leqslant x_\beta$. We then have the following lemma.

Lemma 6.4. *The $F^+(k,i,d,B')$ computed by using the average keyword frequency histogram $\mathcal{H}$ is an upper bound for the keyword frequency f_{k,R_x} of any route $R_x \in \mathcal{C}_{i,d}^{B'}$.*

Proof. We first assume that the optimal route in route set $\mathcal{C}_R$ is R^* and the corresponding sub-route of R^* from vertex v_i to the destination d is R_x^*. We further assume that those edges containing keyword k on sub-route R_x^* are $E_{R_x^*}(k)$=$\{e_i, e_{i+1}, \ldots, e_\alpha\}$. We then have $E_{R_x^*}(k)$⊆$E_{i,d}^{B'}(k)$ since any edge that contains keyword k and is not in $E_{i,d}^{B'}(k)$ will violate the cost budget constraint (cf. Eq. (6.16)).

The frequency of keyword k on route R_x^* is f_{k,R_x^*}=$\sum_{e \in E_{R_x^*}(k)} f_{k,e}$ and the cost of R_x^* satisfies $\lambda(R_x^*) \leqslant B'$. In Figure 6.5, we record those edges on the left side of the cost budget B' or insects with B' as $E_{\mathcal{H}}$=$\{e_1, e_2, \cdots, e_\beta\}$. Then, for any edge e on route R_x^*, if $e \notin E_{\mathcal{H}}$, the average keyword frequency of e cannot be larger than any edge in $E_{\mathcal{H}}$. Therefore, the $F^+(k,i,d,B')$ computed based on $E_{\mathcal{H}}$ is an upper bound for the keyword frequency f_{k,R_x} of any route $R_x \in \mathcal{C}_{i,d}^{B'}$. □

Example 6.6. *Continued with Example 6.5, the costs, keyword frequencies, and average keyword frequencies of these edges in $E_{i,d}^{B'}(k)$ are summarized in Table 6.3. The corresponding average keyword frequency histogram is illustrated in Figure 6.6.*

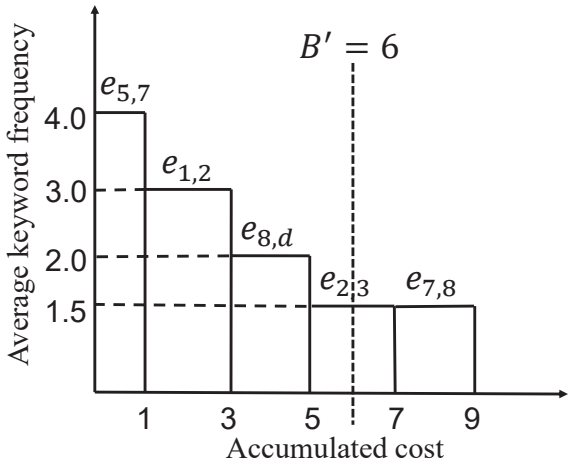


Figure 6.6: Average keyword frequency histogram for Figure 6.4

Table 6.3: The edges $E_{i,d}^{B'}(k)$ containing keyword k in Figure 6.4

Edge	**Cost**	**Keyword frequency**	**Average keyword frequency**
$e_{1,2}$	2	6	3.0
$e_{2,3}$	2	3	1.5
$e_{5,7}$	1	4	4.0
$e_{7,8}$	2	3	1.5
$e_{8,d}$	2	4	2.0

With Eq. (6.17), we have

$$\begin{aligned} F^+(k,i,d,B') &= f_{k,e_{5,7}} + f_{k,e_{1,2}} + f_{k,e_{8,d}} + \bar{f}_{k,e_{2,3}} \cdot (B' - 5) \\ &= 4 + 6 + 4 + 1.5 \cdot (6 - 5) \\ &= 15.5 \end{aligned}$$

6.4.5 Query Processing Algorithm

Algorithm 6.3 presents the algorithm for the exact solution. The initialization and processing procedures are elaborated below.

Initialization: Initially, a max-priority queue U is created to store partial routes during the query processing and a partial route R with the start location s is enqueued (lines 1-3). Meanwhile, route R^* stores the optimal route and is initialized to the shortest route from s to d, i.e., $SR_{s,d}$ (line 4). Note that R^* can be also initialized

by other heuristic methods without modifying the algorithm. In addition, a reverse Dijkstra's algorithm is conducted to compute the shortest cost of each vertex to the destination d (line 5).

Query processing: In each iteration, the partial route R with the largest upper bound score is dequeued from U (line 7). If the upper bound score of R is not larger than that of the current optimal route R^*, R is pruned (line 8). Otherwise, the end vertex v_i of R is computed (line 9). Each adjacent vertex v_j of v_i is then concatenated to R to generate a longer route R' if v_j has not been visited by R (lines 10-11). R' will be pruned if it is impossible to generate a candidate route (cost pruning, line 12). If R' survives from the cost pruning, arrives at the destination d and has a larger score than the current optimal route R^*, it is used to update R^* (lines 13-14). If R' is just a new partial route, its upper bound score $\tau^+(R')$ is computed (lines 15-16). If $\tau^+(R')$ is not larger than that of the current optimal route, R' is pruned (line 17). Otherwise, R' is add to the priority queue U for further exploration (line 18). The algorithm terminates when U is empty (line 6) and the optimal route R^* is returned (line 19).

6.4.6 Complexity Analysis

We assume that the minimum cost among all edges is c_{min} and the maximum vertex degree is D. In the worst case, Algorithm 6.3 needs to check $O(D^{\frac{B}{c_{min}}})$ routes. Therefore, the efficiency of pruning methods greatly affects the practical time complexity.

6.5 Time-Bounded Solution

In some applications, users would like to impose a constraint on the response time to avoid waiting for a long time. To meet this requirement, we propose time-bounded solution (TBS) which returns query results within the user-specified response time

Algorithm 6.3 The exact solution

Input: BCIR query q=$(s, d, \mathcal{K}_q, B)$
Output: The optimal route R^*

```
1: Initialize U ← an empty priority queue
2: Initialize R ← ⟨s⟩
3: U.enqueue(R, 0)
4: R* ← SR_{s,d}
5: Compute λ(SR_{i,d}) for each v_i by reverse Dijkstra's algorithm
6: while U is not empty do
7:     R ← U.dequeue()
8:     if τ+(R) ⩾ τ(R*) then                                  ▷ score pruning
9:         v_i ←endVertex(R)
10:        for v_j ∈ adj(v_i) do
11:            R' ← R ⊕ {v_j}
12:            if λ(R') + λ(SR_{j,d}) ⩽ B then                ▷ cost pruning
13:                if (v_j = d) and (τ(R') > τ(R*)) then
14:                    R* ← R'                                ▷ update the optimal route
15:                else
16:                    Compute upper bound score τ+(R')
17:                    if τ+(R') > τ(R*) then                 ▷ score pruning
18:                        U.enqueue(R', λ(R'))
19: return R*
```

limit.

The idea of TBS is based on the observation that candidate route set $\mathcal{C}^B_{s,d}$ between two query locations s and d often share edges with each other. Therefore, it is possible to convert one route to another by changing their sub-routes. One candidate route can be enhanced if we can replace its sub-routes with other sub-routes such that the new candidate route has a better score.

Figure 6.7 illustrates the framework for TBS which takes as input the road network, the BCIR query and a response time limit, and returns an approximate query result. In TBS, we have three major components:

- **routes initialization** generates a set of candidate routes;
- **inferior sub-route identification** randomly selects one candidate route from the generated candidate route set and identifies an inferior sub-route with limited contribution to the score of the selected candidate route for enhancement;

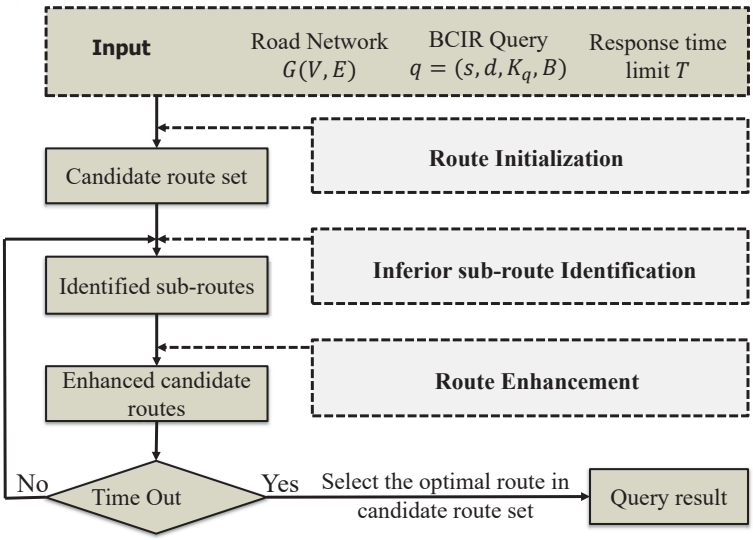


Figure 6.7: Framework of time-bounded solution

- **route enhancement** generates a new sub-route to replace the identified inferior sub-route and enhances the selected candidate route.

When response time limit permits, inferior sub-route identification and route enhancement repeat continuously. Finally, the optimal route in the candidate route set is returned as the query result. The details of the three components are elaborated in the subsequent sub-sections.

6.5.1 Routes Initialization

Route initialization generates a set of h candidate routes $\mathcal{C}$ so that time-bounded solution can further enhances these h candidate routes continuously until reaching the response limit.

The reason for generating h candidate routes rather than one is twofold. On the one hand, it is of high probability for one candidate route to get stuck in a local optimum, thus reducing the probability of obtaining the optimal route. On the other hand, enhancing h candidate routes is faster to approach the optimal route than enhancing one. As illustrated in Figure 6.8 where the grey ellipse is the whole

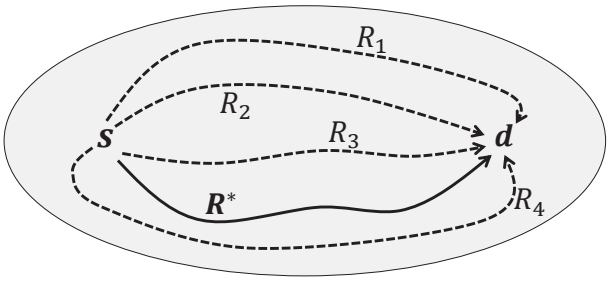


Figure 6.8: Initialized candidate routes and the search space

search space, if only route R_1 is generated, it will be a long way to enhance R_1 to get the optimal route R^*. In contrast, if four routes R_1, R_2, R_3 and R_4 are generated, it should be very fast to enhance R_4 to get the optimal route R^*.

The setting of h concerns two aspects. On the one hand, though a large h could have a good coverage of the search space, it also requires much time to generate and enhance the candidate route set. On the other hand, a small h cannot reflect the advantage of route set initialization since it only covers a small search space. Therefore, we will tune h in the experiments to determine its appropriate value.

To generate h candidate routes that are distributed randomly over the whole search space, we propose two methods below.

A. Random route sampling

Random route sampling samples a candidate route by gradually expands edges from the start location s. Initially, a route R with the start location s is generated. In each expansion, one more edge is added to the end of R. To ensure that the added edge $e_{i,j}$ satisfies the cost budget, $e_{i,j}$ should satisfy $\lambda(R) + c_{i,j} + \lambda(SR_{j,d}) \leqslant B$, where $c_{i,j}$ is the cost of $e_{i,j}$ and $SR_{j,d}$ is the shortest route from v_j to the destination d. Finally, R reaches the destination d and is returned as a candidate route.

Example 6.7. *Take the BCIR query in Figure 6.2 for example. Initially, we have* $R{=}\langle s\rangle$. *There are three edges adjacent to* s, *i.e.,* $e_{s,1}$, $e_{s,2}$ *and* $e_{s,3}$. *We randomly select one edge from the three edges and assume* $e_{s,2}$ *is selected, thus* $R{=}\langle s, v_2\rangle$.

Next, we consider the two adjacent edges $e_{2,1}$ and $e_{2,d}$ of the end vertex v_2 of R. Since $\lambda(R) + c_{2,1} + \lambda(SR_{1,d}) = 5 + 5 + 5 > B = 12$, only $e_{2,d}$ is feasible. Therefore, we have R=$\langle s, v_2, d\rangle$ which reaches the destination d, thus a candidate route.

B. Route sampling by adapted nearest neighbour heuristics

Since random route sampling generates candidate routes without considering the query keywords on edges, the initialized candidate routes may have very small scores. In this case, it will take a long time to enhance the candidate routes. To deal with this issue, we propose to sample candidate routes by adapted nearest neighbours (NN) heuristics. Traditional NN heuristics method [27] starts from the start query location and gradually adds the nearest edge containing query keywords until reaching the destination. In this way, we can always generate one candidate route. However, time-bounded solution requires h candidate routes. Therefore, we adapt traditional NN heuristics method so as to generate h candidate routes.

The initialization of adapted NN heuristics method is the same as random sampling. In each expansion, we compute the h nearest edges E^h containing query keywords for the end vertex v_i of R. Then, one edge is randomly selected from E^h and added to the end of R. The remaining issue is how to compute the h nearest edges for vertex v_i. For each BCIR query q, we compute all those edges, E_q, that contain at least one query keyword in $\mathcal{K}_q$ and satisfy the cost budget B, i.e.,

$$E_q = \{e_{f,g} | e_{f,g} \in E \wedge (\mathcal{K}_{f,g} \cap \mathcal{K}_q \neq \varnothing) \wedge \Gamma(s, f, g, d) \leqslant B\} \tag{6.18}$$

where function $\Gamma(s, f, g, d)$=$\lambda(SR_{s,f}) + c_{f,g} + \lambda(SR_{g,d})$ is the same as that in Eq. (6.16). For ease of discussion, we call E_q positive edges. Then, we select the h nearest edges E^h for vertex v_i from E_q.

Example 6.8. *We still take the BCIR query in Figure 6.2 for example and initialize R=$\langle s\rangle$. First, we have positive edges E_q=$\{e_{s,1}, e_{s,3}, e_{1,2}, e_{1,d}\}$. If h=2, we have*

E^h=$\{e_{s,1}, e_{s,3}\}$. Assuming that $e_{s,3}$ is selected, we have R=$\langle s, v_3\rangle$. Then we consider the two nearest edges containing query keywords for v_3. However, we cannot find a feasible edge containing query keywords to add to v_3. Therefore, R goes directly to the destination d and generates a candidate route R=$\langle s, v_3, d\rangle$.

6.5.2 Inferior Sub-route Identification

After generating h candidate routes $\mathcal{C}$, the next step is to refine these h candidate routes until reaching the response time limit T. Each time, a candidate route R is randomly selected from $\mathcal{C}$. The goal of inferior sub-route identification is to find a sub-route of R for enhancement.

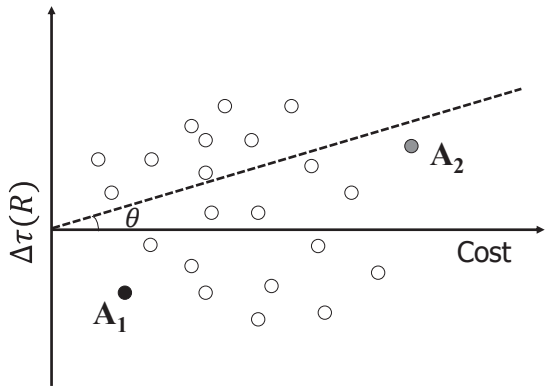


Figure 6.9: The distribution of all sub-routes of R

Figure 6.9 illustrates the distribution of all the sub-routes of R with respect to their costs and $\Delta\tau(R)$ values, where $\Delta\tau(R)$ value of sub-route $R_{i,j}$ is computed as below.

$$\Delta\tau(R) = \tau(R) - \tau(R - R_{i,j}) \tag{6.19}$$

where $\tau(R - R_{i,j})$ computes the route score after removing sub-route $R_{i,j}$ from R. Obviously, $\Delta\tau(R)$ quantifies the change of route score after removing the sub-route $R_{i,j}$.

Since it is difficult to bound the value of $\frac{\Delta\tau(R)}{\lambda(R_{i,j})}$, we compute the corresponding

angle $\theta(R_{i,j})$ in Figure 6.9 by

$$\theta(R_{i,j}) = \arcsin \frac{\Delta\tau(R)}{\lambda(R_{i,j})} \tag{6.20}$$

Angle $\theta(R_{i,j})$ quantifies the ratio between $\Delta\tau(R)$ and the cost of sub-route $R_{i,j}$. According to Eq. (6.20), we have $\theta(R_{i,j}) \in (-90^\circ, 90^\circ)$ since $\lambda(R_{i,j})$ is always positive while $\Delta\tau(R)$ could be both positive and negative.

Intuitively, to increase the score of candidate route R, we need to replace those sub-routes whose costs are large while $\Delta\tau(R)$ values are small, i.e., sub-routes with small θ. To this end, we devise two methods as below.

A. Ranking-based inferior sub-route identification

Ranking-based identification orders all sub-routes according to the θ value. First, the sub-route with the minimum θ value is selected. If we can replace this sub-route with a better one, this sub-route is identified. Otherwise, the second sub-route is checked. ranking-based identification repeats this operation until one sub-route is identified. For example, In Figure 6.9, ranking-based inferior sub-route identification will first select the sub-route corresponding to the black point A_1 since it has the smallest angle θ.

B. Threshold-based inferior sub-route identification

In threshold-based inferior identification, we set a threshold ϕ for θ and compute the longest sub-route $R_{i,j}$ of R such that

$$R_{i,j} = \arg \max_{R_{i,j} \in R \wedge \theta(R_{i,j}) \leqslant \phi} \lambda(R_{i,j}) \tag{6.21}$$

In Figure 6.9, assuming that the threshold is the dashed line, threshold-based sub-route identification will select the sub-route corresponding to the grey point A_2 since it locates below the threshold line and has the largest cost. During the query

processing, we first set ϕ=-80° and increase it by 10° gradually until a sub-route is identified.

6.5.3 Route Enhancement

After identifying the inferior sub-route $R_{i,j}$ of candidate route R, we need to find a new sub-route to replace $R_{i,j}$ such that the new candidate route is better than R. Actually, any route between v_i and v_j such that its cost is less than B'=$B-(\lambda(R)-\lambda(R_{i,j}))$ can be used to replace $R_{i,j}$. However, it is computationally prohibitive to evaluate all these routes to select the optimal one to replace $R_{i,j}$.

Intuitively, adding those edges containing query keywords to an existing candidate route is of high probability to increase the route score. Therefore, given sub-route $R_{i,j}$ of candidate route R, we compute a candidate route set $\mathcal{C}'$ by evaluating all the positive edges in E_q.

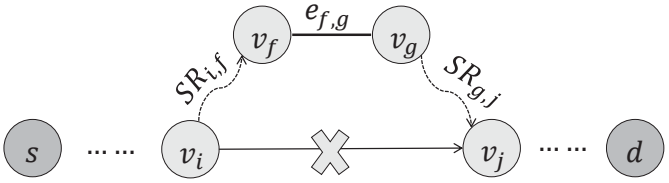


Figure 6.10: Route enhancement in TBS.

For each edge $e_{f,g}$∈E_q, according to Figure 6.10, the new candidate route generated by replacing sub-route $R_{i,j}$ with respect to edge $e_{f,g}$ is

$$R_{new} = R_{s,i} \oplus SR_{i,f} \oplus e_{f,g} \oplus SR_{g,j} \oplus R_{j,d} \tag{6.22}$$

where $R_{s,i}$ is the sub-route (from s to v_i) of R, $R_{j,d}$ is the sub-route (from v_j to d) of R, $SR_{i,f}$ is the shortest route from vertex v_i to vertex v_f, and $SR_{g,j}$ is the shortest route from vertex v_g to vertex v_j.

The algorithm for computing the candidate route set $\mathcal{C}'$ is presented in Algorithm 6.4 which receives as input a candidate route R with its sub-route $R_{i,j}$, and

the positive edges E_q. Initially, an empty route set $\mathcal{C}'$ is created to store potential candidate routes (line 1). In each iteration, one edge from E_q that is not in R is evaluated (line 2), and a new route R_{new} is computed (line 3). The new route R_{new} may contain duplicate edges since the two shortest routes, $SR_{i,f}$ and $SR_{g,j}$, may contain edges in sub-routes $R_{s,i}$ and $R_{j,d}$. After removing duplicate edges (line 4), R_{new} is added to the candidate route set $\mathcal{C}'$ if $\lambda(R_{new}) \leqslant B$ and $\tau(R_{new}) > \tau(R)$(lines 5-6). By evaluating all the edges in E_q, algorithm generates a set of candidate routes $\mathcal{C}'$ whose scores are larger than R (line 7).

Algorithm 6.4 Enhance_route(R, $R_{i,j}$, E_q)

Input: Candidate route R and its sub-route $R_{i,j}$, and positive edges E_q
Output: A set of new candidate route $\mathcal{C}'$

1: $\mathcal{C}' \leftarrow \varnothing$ ▷ Store the candidate routes
2: **for** $e_{f,g} \in E_q - E_R$ **do**
3: $R_{new} \leftarrow R_{s,i} \oplus SR_{i,f} \oplus e_{f,g} \oplus SR_{g,j} \oplus R_{j,d}$
4: remove duplicate edges of R_{new}
5: **if** $\lambda(R_{new}) \leqslant B$ and $\tau(R_{new}) > \tau(R)$ **then**
6: add R_{new} to $\mathcal{C}'$
7: **return** $\mathcal{C}'$

With the generated candidate routes $\mathcal{C}'$, we have two methods to update the original candidate route R.

A. Score-aware enhancement

In score-aware enhancement, the optimal route in $\mathcal{C}'$ is selected to replace R in $\mathcal{C}$.

B. Randomness-aware enhancement

Since score-aware enhancement only considers the optimal route in $\mathcal{C}'$, it is of high probability to get the local optimum. Therefore, we introduce randomness-aware enhancement in which one candidate route is randomly selected from $\mathcal{C}'$ to replace the original candidate route R. Though randomness-aware enhancement needs more time than score-aware enhancement to converge, it will gradually improve the quality of the current optimal route since only those routes better than the current optimal route are selected as candidate routes.

6.6 Error-Bounded Solution

In some cases, users would like to sacrifice the quality of query answer to reduce processing time as long as the worst case is under control, i.e., the approximation error of the query answer is bounded. Given an approximate route $\hat{R}$, the corresponding approximate error ϵ is computed as below.

$$\epsilon = \frac{\tau(R^*) - \tau(\hat{R})}{\tau(R^*)} \tag{6.23}$$

where R^* is the optimal route. Obviously, we have $\epsilon \in [0, 1]$.

Formally, an error-bounded solution (EBS) returns approximate query results such that the given approximation error threshold is guaranteed. In the exact solution, we can relax the upper bound score based on the approximation error ϵ to search an approximate route rather than the optimal one, i.e., the following lemma.

Lemma 6.5 (Relaxed score pruning)**.** *Given partial route R and the current optimal route R_c^*, if $(1-\epsilon)\cdot\tau^+(R)\leqslant\tau(R_c^*)$, R can be pruned while guaranteeing that the approximation error is at most ϵ.*

Proof. For each candidate route $R'\in\mathcal{C}_R$ generated by expanding partial route R, we have $\tau(R')\leqslant\ \tau^+(R)$. With the assumption that $(1-\epsilon)\cdot\tau^+(R)\leqslant\tau(R_c^*)$, we have $(1-\epsilon)\cdot\tau(R')\leqslant\tau(R_c^*)$. Assuming that the optimal route is R^*, R^* will be pruned if and only if there exists a route R_c^* such that $(1-\epsilon)\cdot\tau(R^*)\leqslant\tau(R_c^*)$. Accordingly, we have $\tau(R^*) - \epsilon\cdot\tau(R^*) \leqslant \tau(R_c^*)$, i.e., $\frac{\tau(R^*)-\tau(R_c^*)}{\tau(R^*)}\leqslant\epsilon$. Therefore, the approximation error for route R_c^* is at most ϵ. □

Therefore, we can adapt the exact solution to an error-bounded solution by using the relaxed score pruning. In addition, we also employ the time-bounded solution to improve the efficiency of EBS. In Algorithm 6.3, we call a time-bounded solution with a time limit T after Line 4. By doing this, we can obtain a good route efficiently,

thus improving the score pruning. The setting of time limit T will be discussed in the experiments and we call this revised EBS as EBS-T.

Though EBS can further reduce the number of iterations in Algorithm 6.3 by using a tighter score bound, it still requires a high time cost to compute the final query results if the error ratio is small and the cost budget is large, which will be evaluated in the experiments.

6.7 Experiments

6.7.1 Setups

We evaluate the performance of proposed solutions over three datasets, i.e., the road networks of New York (NY), California (CA) and United Kingdom (UK), where NY dataset is the default one. All datasets are extracted from OpenStreetMap (OSM)[1] and their details are summarized in Table 6.4. The text descriptions of edges are extracted from the text descriptions of those POIs residing on them. Distance is used as the travel cost in the experiments.

The settings of those parameters in the experiments are listed in Table 6.5, where the default values are highlighted in bold. Particularly, the cost budget of each query is specified by a deviation ratio $\mu \geqslant 0$ from the shortest route, i.e., $B{=}(1+\mu) \cdot \lambda(SR_{s,d})$. We randomly generate 50 queries for each setting and compute the average value of the corresponding results for plotting. All algorithms were implemented in Java and run on a PC equipped with Intel(R) Core(TM) i3-2100 CPU @3.10GHz, 8 GB RAM.

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

Table 6.4: Datasets for evaluating BCIR query

Dataset	#vertices	#edges	average #keywords
New York (NY)	6, 393	13, 885	4.25
California (CA)	90, 870	202, 250	2.46
United Kingdom (UK)	338, 838	738, 610	2.65

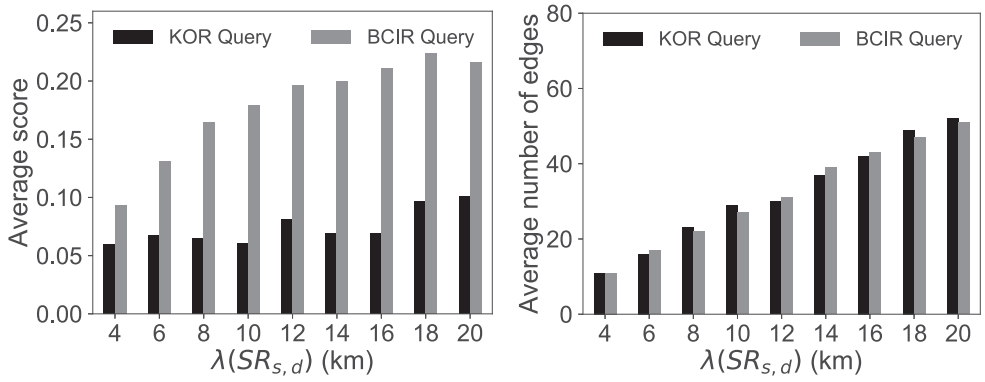
Table 6.5: Parameter setting in BCIR query

Parameter	Meaning	Value
$\|\mathcal{K}_q\|$	the number of query keywords	1, **2**, 3, 4, 5
$\lambda(SR_{s,d})$	the shortest distance (km) from s to d	4, 6, 8, **10**, 12, 14, 16, 18, 20
μ	the deviation ratio	0.05, 0.10, **0.15**, 0.20, 0.25
h	the number of initialized candidate routes	1, 3, 5, 9, **11**, 13, 15, 17, 19

6.7.2 Experimental Results

A. BCIR query vs. KOR query

In order to compare KOR query [8] and BCIR query, we need to adapt KOR query. The cost budget is the same for KOR query and BCIR query. For each edge $e_{i,j} \in E$, we set the objective score in KOR query as $OS(e_{i,j}) = \frac{|\mathcal{K}_{i,j}|}{c_{i,j}}$, where $|\mathcal{K}_{i,j}|$ and $c_{i,j}$ are the number of keywords and cost of $e_{i,j}$, respectively. Then, the adapted KOR query computes the optimal route such 1) it covers the given query keywords, 2) its cost is less than the cost budget, and 3) it has the maximum objective score. Differently, BCIR query is targeted at computing the optimal route within the cost budget such that it is most textually relevant to the query keywords. Figure 6.11(a) shows the scores (i.e., textual relevance) of the routes returned by KOR query and BCIR query. The route scores of KOR query are much smaller than that of BCIR query. Therefore, existing routing applications using keyword coverage query cannot effectively find the optimal route that is most textually relevant to the query keywords. In addition, Figure 6.11(b) presents the number of edges in returned routes for two queries. With the increase of shortest distance, both queries return routes with more edges. However, there is no obvious difference between the sizes of edges for two queries.

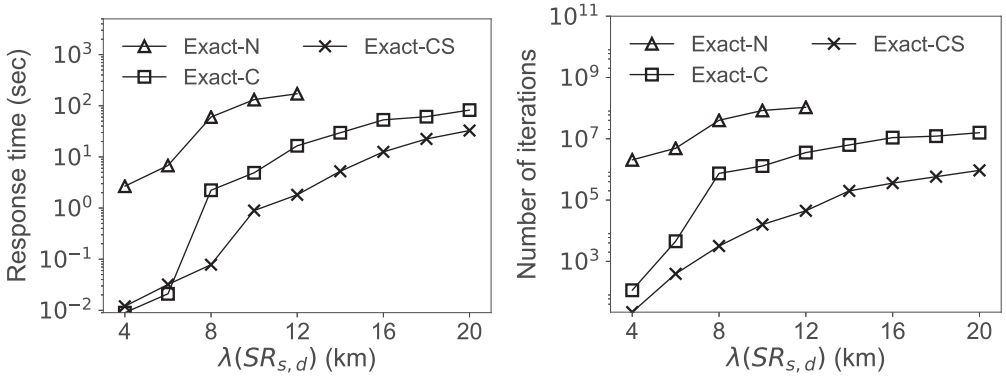


(a) Score (b) Number of edges

Figure 6.11: The route score and number of edges while varying the shortest distance $\lambda(SR_{s,d})$ between two query locations in BCIR query and KOR query, where cost budget $B{=}(1+0.15)\cdot\lambda(SR_{s,d})$

B. Exact Solution

Pruning effectiveness: The exact solution exploits cost pruning and score pruning to reduce the search space. Figure 6.12 shows the response time and the number of iterations in Algorithm 6.1 while using no pruning (Exact-N), cost pruning (Exact-C) and cost-score pruning (Exact-CS), where Exact-N does not involve any pruning techniques, Exact-C employs the cost pruning, and Exact-CS leverages both cost pruning and score pruning. Note that, we set the upper bound running times of Exact-N and Exact-C to 200 seconds since the real running times are much longer than 200 seconds when the cost budget is large. As suggested by Figure 6.12, the cost pruning reduces iterations by several orders of magnitude and the score pruning can further reduce the number of iterations by around one order of magnitude. After applying both cost and score pruning methods, the number of iterations is greatly reduced, thus making the algorithm efficient for BCIR queries of small cost budgets. In addition, Figure 6.12 indicates that when the shortest distance between two query locations is less than 15 km, the exact solution can return the query results within seconds.



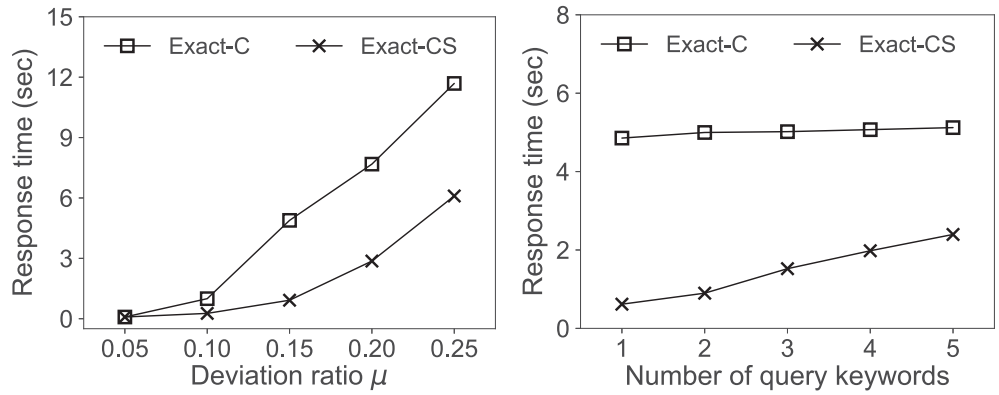
(a) Response time (b) Number of iterations

Figure 6.12: The results while varying the shortest distance $\lambda(SR_{s,d})$, where cost budget $B = (1 + 0.15) \cdot \lambda(SR_{s,d})$

Varying deviation ratio μ: Figure 6.13(a) illustrates the response time of exact solution while varying the cost deviation ratio μ, where the cost of the shortest route is $\lambda(SR_{s,d})$=10 km. With the increase of μ, the response time of Exact-C increases rapidly. In contrast, if both cost pruning and score pruning are used, i.e., Exact-CS,the increase rate is moderate.

Varying the number of query keywords $|\mathcal{K}_q|$: Figure 6.13(b) illustrates the results while varying the number of query keywords. With the increase of the number of query keywords, the response time of Exact-CS gradually increases since it needs more time to compute the upper bound score. Differently, the response time of Exact-C keeps almost the same because it only utilizes cost pruning and does not need to compute upper bound score, thus less dependent on the number of query keywords.

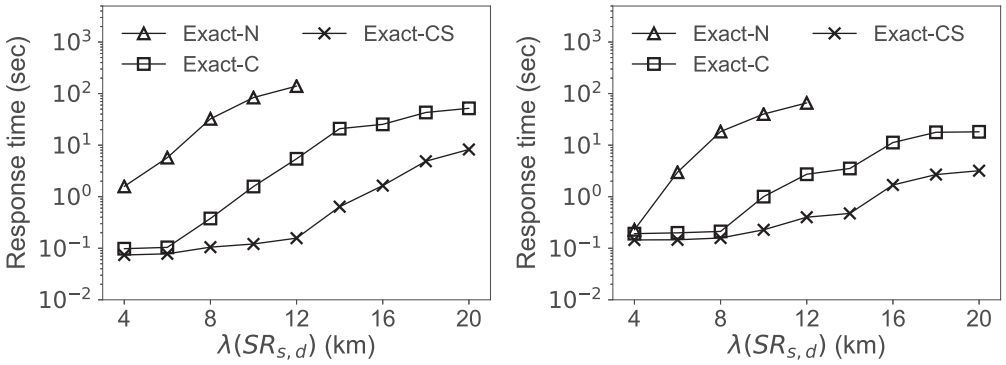
Scalability test: We also evaluate the performance of exact solution on two large datasets, i.e., CA and UK datasets. As illustrated in Figure 6.14, exact solution still performs well when the road network have hundreds of thousands of vertices. In fact, the response time on these two datasets is smaller than that on NY dataset because the query locations are randomly selected and they could be out of city on



(a) Response time (b) Number of iterations

Figure 6.13: The results while varying the deviation ratio μ and number of query keywords

the two large datasets. In general, the density of road network out of city is sparse, thus reducing the number of candidate routes for in BCIR query. In sum, according to Figures 6.12(a) and 6.14, exact solution can process BCIR query within several seconds when the cost budget is less than 15 km.



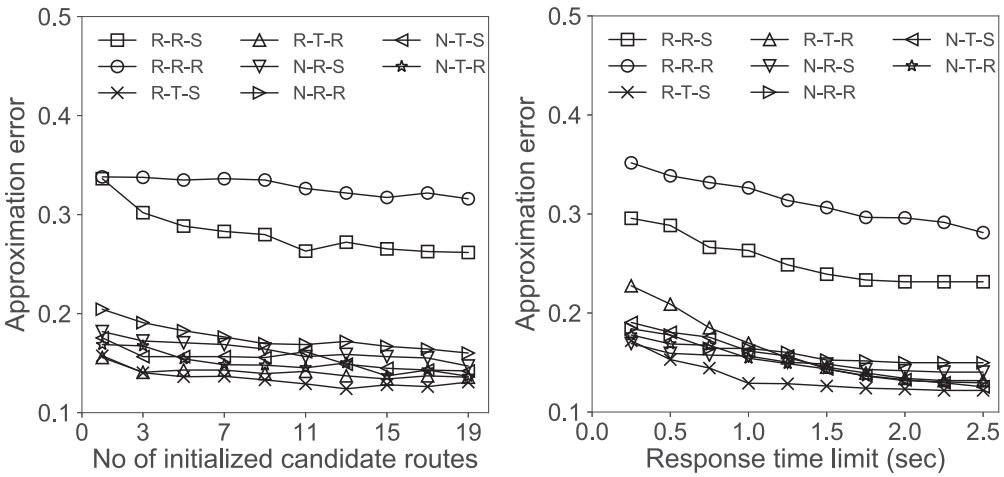
(a) California (b) United Kingdom

Figure 6.14: The response time while varying the shortest distance $\lambda(SR_{s,d})$ on CA and UK datasets

C. Time-Bounded Solution

Method combination comparison: Time-bounded solution (cf. Section 6.5) has three components and each component has two methods, thus 8 combinations in

total. For ease of presentation, we record these combinations as X-Y-Z where X represents routes initialization method, including random route sampling (R) and adapted NN heuristics route sampling (N); Y represents inferior sub-route identification method, including ranking-based identification (R) and threshold-based identification (T); and Z represents enhancement method, including score-aware enhancement (S) and randomness-aware enhancement (R). For example, R-R-S indicates that random route sampling, ranking-based identification and score-aware enhancement are used. Figure 6.15 presents the approximation error (defined in Eq.(6.23)) of returned routes using different combinations. According to Figure 6.15(a), we set the number of initialized candidate routes h=11. Since combination R-T-S performs the best among all the combinations, we only report the results of R-T-S in subsequent plots. In addition, Figure 6.16 illustrates the distribution of response time for all combinations. Obviously, random route sampling consumes less time than adapted NN heuristics route sampling.



(a) Varying h (T=1 sec) (b) Varying response time (h=11)

Figure 6.15: The results of different combinations while varying the number of initialized candidate routes h and the response time limit, where $\lambda(SR_{s,d})$ =15 (km).

GRASP vs. TBS: To demonstrate the performance of time-bounded solution, we

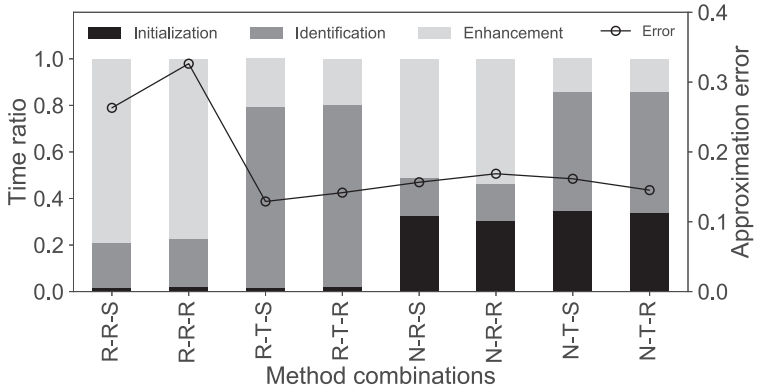


Figure 6.16: The distribution of processing time among three components

adapted the GRASP [76] solution for arc orienteering problem (AOP) to our BCIR query. Similar to TBS, GRASP also gradually refines the current optimal route by generating new candidate routes. However, GRASP generates candidate routes by searching the whole road network and needs to pre-compute the shortest routes for all pairs of vertices. Since it is inapplicable to pre-compute all pair shortest routes for large-scale road networks, we utilize CH index to compute the shortest route between two locations for GRASP online. Figure 6.17 illustrates the results of R-T-S and GRSAP with two large shortest distances, 15 km and 20 km. According to Figure 6.17, R-T-S greatly outperforms GRASP algorithm. The performance of GRASP algorithm is not satisfying because it needs to search the whole road network in each iteration and cannot iterate enough times to improve the quality of query results.

Varying μ and $|\mathcal{K}_q|$: Figure 6.18 presents the results while varying the deviation ratio μ and the number of query keywords $|\mathcal{K}_q|$. With the increase of μ, the approximation error first increases because the search space increases. However, the approximation error then decreases when μ is larger than 0.15 because the increase of the optimal route score slows down. Differently, with the increase of $|\mathcal{K}_q|$, the approximation error always decreases. According to Figure 6.18, R-T-S outperforms

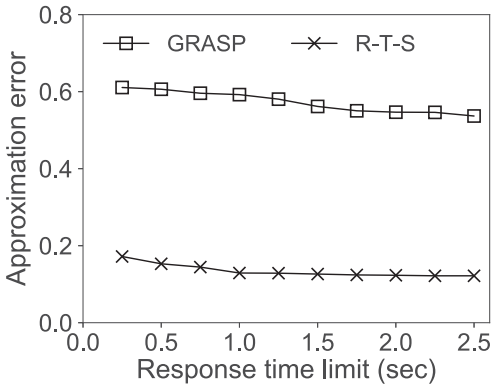
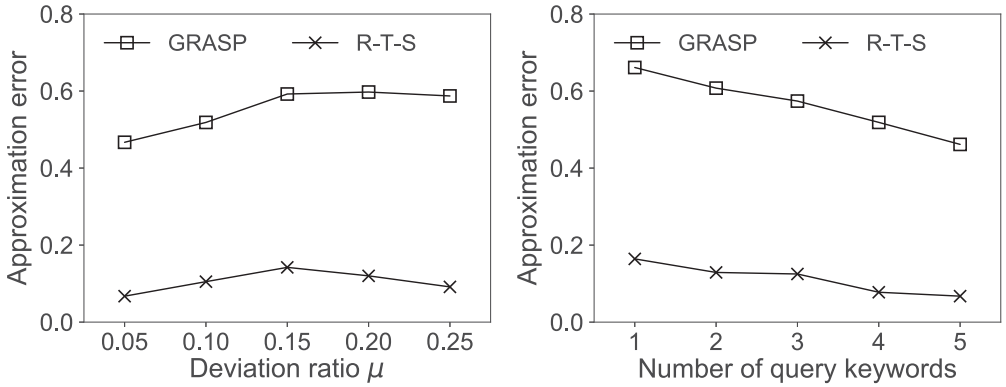


Figure 6.17: The results of TBS and GRASP while varying the processing time limit where cost budget $B = (1 + 0.15)*15$ (km).

GRASP in all cases.



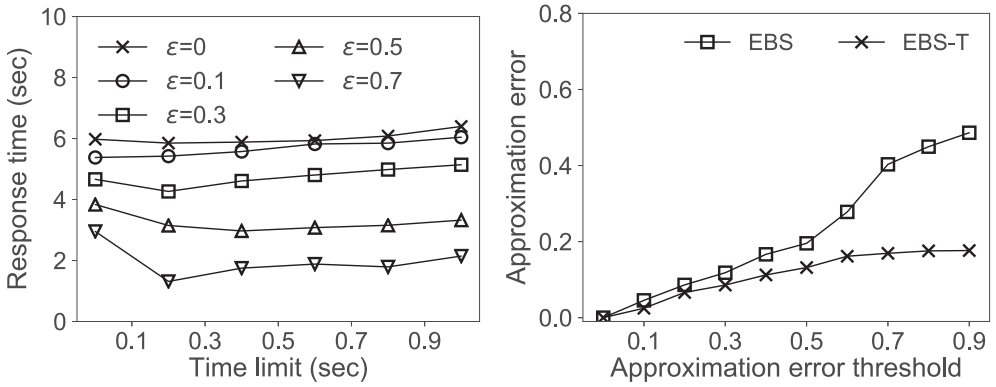
(a) Varying μ (b) Varying $|\mathcal{K}_q|$

Figure 6.18: The results while varying the deviation ratio μ and the number of query keywords $|\mathcal{K}_q|$, where cost budget $B = (1 + 0.15)*15$ (km).

D. Error-Bounded Solution

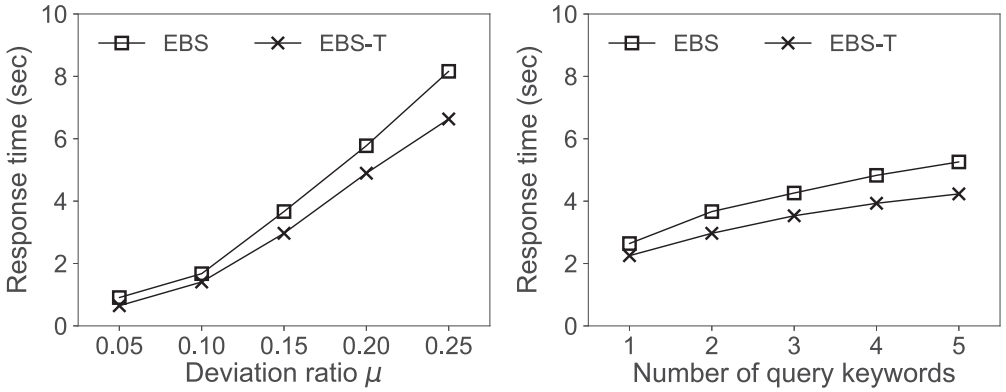
Figure 6.19(a) illustrates the response time while varying the time limit T of time-bounded solution in EBS-T. According to Figure 6.19(a), the response time reduces most when setting T=0.2 second. Figure 6.19(b) presents the approximation error of EBS and EBS-T (T=0.2 second) while varying the approximation error threshold ϵ. With the increase of ϵ, the approximation errors of both solutions increase. However,

the increase ratio of EBS-T is much smaller than that of EBS. In addition, we also evaluate the performance of EBS and EBS-T while varying the deviation ratio μ and the number of query keywords. As suggested by Figure 6.20, the response time will increase with the increase of μ and $|\mathcal{K}_q|$ and EBS-T performs better than EBS.



(a) Varying time limit T (b) Varying ϵ

Figure 6.19: The response time while varying time limit T and the approximation error while varying the approximation error threshold ϵ, where cost budget B=(1 + 0.15)∗15 (km)



(a) Varying μ (b) Varying $|\mathcal{K}_q|$

Figure 6.20: The response time while varying the deviation ratio μ and the number of query keywords $|\mathcal{K}_q|$, where $\epsilon = 0.5$ and cost budget B=(1 + 0.15)∗15 (km)

6.8 Summary

In this chapter, we propose the BCIR query to retrieve the route that is most textually relevant to the user-specified query keywords within a travel cost budget. To efficiently process BCIR queries, we propose an exact solution with effective pruning techniques and two approximate solutions regarding the response time and the quality of query results, respectively. As demonstrated via extensive experiments, the proposed solutions achieve satisfying performance over different datasets. BCIR provides a new style of route query that can be applied in different applications ranging from route planning to location-aware recommendation. The future extensions of BCIR query may include 1) searching the top-k optimal routes rather than the optimal one, 2) finding a BCIR route that allows duplicate vertices or edges, 3) removing the query location constraints and retrieving the optimal route over the whole road network.

Chapter 7

Probabilistic Time-constrained Route Query

7.1 Introduction

With the increasing popularity of GPS-enabled mobile devices, computing a route over road networks to pass required service providers becomes a fundamental operation in many location-based applications and location-aware recommendation systems [100]. Figure 7.1 illustrates a small road network on which edges and vertices represent road segments and intersection points of road segments, respectively. In addition, a set of POIs, $o_1 \sim o_6$, such as restaurants, banks and supermarkets (represented by white circles) reside on road segments and provide certain services described by keywords (k_i). A typical route query is to find routes between given start location and destination that pass through a number of user-specified POIs on the road network with some criteria (e.g., small travel distance or time). Assuming that v_4 is the current location of a user who has a destination of v_5, one example of route query is: "*Give me a driving route from v_4 to v_5 that sequentially passes a bank offering exchange (k_1, k_2) service and a supermarket (k_6) with the minimum travel time*".

While many existing studies [8, 51, 73, 95] on route query assume that the traffic

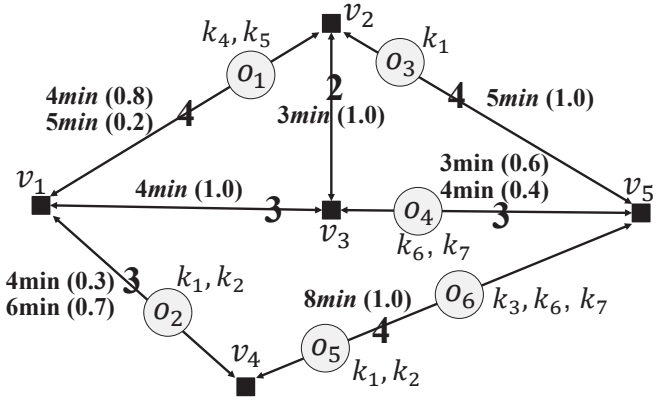


Figure 7.1: An example of uncertain road network, where circles are POIs with keyword descriptions. The length and travel time of each edge are labeled beside. For example, the length of edge $e_{1,2}$=(v_1, v_2) is 4 and the travel time is 4 minutes in a probability of 0.8 and 5 minutes in a probability of 0.2. In addition, an example of PTR query is q=(v_4,v_5, 12:00, $\{\langle k_1, k_2 : 30min\rangle, \langle k_6 : 20min\rangle\}, 1, 0.5$) which finds the optimal routes from v_4 to v_5 to cover keywords "k_1, k_2" and "k_6" sequentially.

on road networks is deterministic. However, this is not the real scenario. In practice, vehicle speeds collected on roads are uncertain and imprecise since different vehicles may have various possible speeds on the same road. Although there exist some studies on route query on uncertain road network [41, 65, 43, 14, 94], they only compute probabilistic routes with the optimal travel time between two locations and do not consider retrieving routes to cover POIs.

In addition, it is often necessary to consider the time constraints, i.e., the service time, of POIs. For example, in Figure 7.1, one may want to find a route from v_4 to v_5, and drop by a bank (k_1) for some exchange (k_2) on the route. Starting from v_4, we consider two routes R_1=(v_4, o_2, v_1, v_3, v_5) and R_2=(v_4, o_5, v_5) that cover o_2 and o_5, respectively. We assume that the departure time is 12:00 and the business hours of o_2 and o_5 are {9:00-13:00, 14:00-18:00} and {8:00-12:00, 13:00-17:00}, respectively. Without considering the service time constraints of banks, o_5 is better than o_2 since R_2 is much shorter than R_1. However, one will find that o_5 is closed (the earliest arrival time is 12:02 which will be discussed in Example 7.4) if s/he chooses R_2. In

this case, bank o_2 is the better choice. Therefore, it is important to consider the service time constraints of POIs for route query.

Inspired by the practical requirements of route query over road network above, we model road network as *uncertain road network* (URN) over which the travel time of each edge is captured by a set of travel time samples. Moreover, based on uncertain road network, we propose the *probabilistic time-constrained route* (PTR) query to find the best routes that sequentially pass through a number of POIs containing user-specified query keywords (e.g., types of services provided by POIs), satisfy the service time constraints of POIs, and have small travel times with high confidence.

PTR query is particularly useful in location-based services. When touring a city, a tourist may be unfamiliar with the service hours of POIs (the exhaustive manual checking is boring) and the traffic condition there. In this case, PTR query can help the tourist find the routes that satisfy both keyword and service time constraints of POIs, and have the optimal travel times with high confidence.

However, efficient answering of PTR queries is rather challenging. Actually, as it will be discussed in Section 7.2.3, PTR query is an NP-hard problem. In order to tackle the PTR problem, we design effective pruning strategies to filter out false alarms and obtain a small set of candidate routes. After that, a refinement step based on sampling is conducted over the candidate routes to get the final query results.

7.2 Problem Statement

7.2.1 Data Models

A. Uncertain road network

Uncertain road network model captures the inherent uncertainty of the travel times over road network and is formally defined as below.

Definition 7.1. *(Uncertain Road Network, URN) An uncertain road network is*

denoted by $\mathcal{G}$=$(V(\mathcal{G}), E(\mathcal{G}))$, *where*

- $V(\mathcal{G})$ *is a set of vertices and each vertex* $v_i \in V(\mathcal{G})$ *represents an intersection of roads or a road terminal,*
- $E(\mathcal{G})$ *is a set of directed edges* $e_{i,j}$ *(i.e., the road segment from vertex* v_i *to vertex* v_j*) with length* $|e_{i,j}|$*, each associated with a random variable* $\boldsymbol{t}(e_{i,j})$ *representing the uncertain times for vehicles to travel from* v_i *to* v_j*,*

where the distribution $\boldsymbol{t}(e_{i,j})$ *is captured by* T *discrete random samples collected on edge* $e_{i,j}$*.*

Example 7.1. *Figure 7.1 illustrates an example of uncertain road network with 5 vertices and 7 edges. The length and travel time of each edge are marked beside. In particular, the travel time of edge* $e_{1,2}$ *is 4 minutes with a probability of 0.8 and 5 minutes with a probability of 0.2, which means 80% of the* T *travel time samples are 4 minutes while 20% are 5 minutes. For simplicity, in this example, only three edges are uncertain and the others are assumed to be deterministic.*

In this work, we assume that edges $e_{i,j}$ and $e_{j,i}$ have the same travel time, i.e., $\mathbf{t}(e_{i,j})$=$\mathbf{t}(e_{j,i})$. Nonetheless, our proposed approaches can be also applied to roads with asymmetric travel time. In addition, the travel time samples for adjacent edges are collected independently in the same time period as discussed in Section 7.3.6. Thus, in some ways, the correlation between adjacent edges has been implicitly considered with respect to the temporal relationship.

Obviously, the travel time $\mathbf{t}(e_{i,j})$ on edge $e_{i,j}$ is bounded by $[t^-(e_{i,j}), t^+(e_{i,j})]$, where $t^-(e_{i,j})$=$\min_{t \in \mathbf{t}(e_{i,j})} t$ and $t^+(e_{i,j})$=$\max_{t \in \mathbf{t}(e_{i,j})} t$. Given a route R=(v_1, v_2, ..., v_l), its travel time over uncertain road network $\mathcal{G}$ is

$$\mathbf{t}(R) = \sum_{i=1}^{l-1} \mathbf{t}(e_{i,i+1}). \tag{7.1}$$

The minimum and maximum travel times of R are

$$t^-(R) = \sum_{i=1}^{l-1} t^-(e_{i,i+1}) \tag{7.2}$$

and

$$t^+(R) = \sum_{i=1}^{l-1} t^+(e_{i,i+1}), \tag{7.3}$$

respectively.

B. Possible Worlds of URNs

Possible world [1] semantics is widely used in probabilistic databases, where each possible world is a materialized instance of the database that can appear in the real world. Similarly, the possible worlds of an uncertain road network $\mathcal{G}$ are defined below.

Definition 7.2. *(Possible Worlds of $\mathcal{G}$) A possible world, $w(\mathcal{G})$, of uncertain road network $\mathcal{G}$ is a deterministic graph on which the random variable $\boldsymbol{t}(e_{i,j})$ of each edge $e_{i,j}$ takes a certain value $x(e_{i,j})$ and its appearance probability, $Pr\{w(\mathcal{G})\}$ is:*

$$Pr\{w(\mathcal{G})\} = Pr\left\{ \bigwedge_{\forall e_{i,j} \in E(\mathcal{G})} \boldsymbol{t}(e_{i,j}) = x(e_{i,j}) \right\}. \tag{7.4}$$

The set of all the possible worlds of $\mathcal{G}$ is recorded as $\mathcal{W}(\mathcal{G})$.

In Definition 7.2, the appearance probability of possible world, $Pr\{w(\mathcal{G})\}$, is the joint probability that each variable $\mathbf{t}(e_{i,j})$ takes value $x(e_{i,j})$. Considering the assumption that random variables $\mathbf{t}(e_{i,j})$ are independent, we have

$$Pr\{w(\mathcal{G})\} = \prod_{\forall e_{i,j} \in E(\mathcal{G})} Pr\{\mathbf{t}(e_{i,j}) = x(e_{i,j})\}. \tag{7.5}$$

Therefore, the number of possible worlds of $\mathcal{G}$ is exponential with respect to the number of edges in $\mathcal{G}$, i.e., $\prod_{e_{i,j} \in E} |\mathbf{t}(e_{i,j})|$ ($= T^{|E|}$ in the worst case), where $|\mathbf{t}(e_{i,j})|$ is the number of distinct values in $\mathbf{t}(e_{i,j})$.

Example 7.2. *Table 7.1 presents 8 possible worlds, $w_1(\mathcal{G}) \sim w_8(\mathcal{G})$, of the uncertain road network in Figure 7.1. For example, possible world $w_1(\mathcal{G})$ takes $\boldsymbol{t}(e_{1,2})$=4, $\boldsymbol{t}(e_{1,4})$=4, and $\boldsymbol{t}(e_{3,5})$=3 (the travel times of other edges are deterministic and we can regard that the collected samples have the same value). From Figure 7.1, we have*

$$\begin{aligned} Pr\{w_1(\mathcal{G})\} &= Pr\{\boldsymbol{t}(e_{1,2}) = 4\} \times Pr\{\boldsymbol{t}(e_{1,4}) = 4\} \times Pr\{\boldsymbol{t}(e_{3,5}) = 3\} \\ &= 0.8 \times 0.3 \times 0.6 \\ &= 0.144. \end{aligned}$$

Table 7.1: Possible worlds of the uncertain road network in Figure 7.1

possible world	**the travel time of edges**	$Pr(w_i)$	$\mathcal{R}^1_{w_i}$
$w_1(\mathcal{G})$	$\mathbf{t}(e_{1,2})$=4, $\mathbf{t}(e_{1,4})$=4, $\mathbf{t}(e_{3,5})$=3	0.144	R_1
$w_2(\mathcal{G})$	$\mathbf{t}(e_{1,2})$=4, $\mathbf{t}(e_{1,4})$=4, $\mathbf{t}(e_{3,5})$=6	0.096	R_2
$w_3(\mathcal{G})$	$\mathbf{t}(e_{1,2})$=4, $\mathbf{t}(e_{1,4})$=6, $\mathbf{t}(e_{3,5})$=3	0.336	R_1
$w_4(\mathcal{G})$	$\mathbf{t}(e_{1,2})$=4, $\mathbf{t}(e_{1,4})$=6, $\mathbf{t}(e_{3,5})$=6	0.224	R_2
$w_5(\mathcal{G})$	$\mathbf{t}(e_{1,2})$=5, $\mathbf{t}(e_{1,4})$=4, $\mathbf{t}(e_{3,5})$=3	0.036	R_1
$w_6(\mathcal{G})$	$\mathbf{t}(e_{1,2})$=5, $\mathbf{t}(e_{1,4})$=4, $\mathbf{t}(e_{3,5})$=6	0.024	R_2
$w_7(\mathcal{G})$	$\mathbf{t}(e_{1,2})$=5, $\mathbf{t}(e_{1,4})$=6, $\mathbf{t}(e_{3,5})$=3	0.084	R_1
$w_8(\mathcal{G})$	$\mathbf{t}(e_{1,2})$=5, $\mathbf{t}(e_{1,4})$=6, $\mathbf{t}(e_{3,5})$=6	0.056	R_2

C. Service Time Constrained POI

On an uncertain road network $\mathcal{G}$, there exist many service time constrained POIs that represent all kinds of service providers such as banks and hotels.

Definition 7.3 (Service Time Constrained Points of Interest)**.** *Each service time constrained POI is in the form of o=(l, e, $\mathcal{K}$, I), where*

- *l is the spatial location of o;*

- *$o.e$=(v_i, v_j) is the edge on which o resides or the nearest edge to o, and the distances from o to vertices v_i and v_j are $dist(o.l, v_i)$ and $dist(o.l, v_j)$, respectively;*
- *$\mathcal{K}$ is a set of keywords describing the properties of o or the services provided by o; and*
- *I is a set of time intervals $\{I_1, I_2, \dots\}$ in which o is valid for providing services.*

The whole collection of POIs is denoted by $\mathcal{O}$.

In Definition 7.3, the time interval set I=$\{I_1, I_2, \dots\}$ specifies the service time constraints of POI o on a 24-hour cycle. For simplicity, in this chapter, we use POIs and time constrained POIs interchangeably when the context is clear.

Example 7.3. *Table 7.2 lists the details of the six POIs, $o_1 \sim o_6$, in Figure 7.1. For example, POI o_2 is on edge $o_2.e$=$e_{1,4}$, $dist(o_2, v_1)$=1.5, contains 2 keywords $o_2.\mathcal{K}$=$\{k_1, k_2\}$, and is associated with two time intervals $o_2.I$=$\{$9:00-13:00, 14:00-18:00$\}$, i.e., the valid time intervals of services.*

Table 7.2: The time constrained POIs in Figure 7.1

POIs	$o.e$	$dist(o, v_i)$	$o.\mathcal{K}$	$o.I$
o_1	$e_{1,2}$	$dist(o_1, v_1)$=3.0	$\{k_4, k_5\}$	{11:00-13:00, 17:00-21:00}
o_2	$e_{1,4}$	$dist(o_2, v_1)$=1.5	$\{k_1, k_2\}$	{09:00-13:00, 14:00-18:00}
o_3	$e_{2,5}$	$dist(o_3, v_2)$=1.0	$\{k_1\}$	{09:00-11:00, 13:00-16:00}
o_4	$e_{3,5}$	$dist(o_4, v_3)$=1.0	$\{k_6, k_7\}$	{08:00-20:00}
o_5	$e_{4,5}$	$dist(o_5, v_4)$=1.0	$\{k_1, k_2\}$	{08:00-12:00, 13:00-17:00}
o_6	$e_{4,5}$	$dist(o_6, v_4)$=2.0	$\{k_3, k_6, k_7\}$	{08:00-21:00}

7.2.2 Probabilistic Time-constrained Route Query

A *probabilistic time-constrained route* (PTR) query is denoted by q=(s, d, t_s, $\mathcal{K}$, h, τ), where s and d are the start location and destination, respectively; t_s is the

departure time; $q.\mathcal{K}$ specifies the query keywords along with the staying times at the corresponding POIs containing the query keywords; h is an integer parameter w.r.t. the number of the returned routes; and $\tau \in (0, 1]$ is a probability threshold to ensure that the returned routes have the minimum travel time in high confidence. Particularly, we have

$$q.\mathcal{K} = \{\langle \mathcal{K}_1, t_1 \rangle, ..., \langle \mathcal{K}_\psi, t_\psi \rangle\} \tag{7.6}$$

where $\mathcal{K}_i$ and t_i $(i = 1, \ldots, \psi)$ are the query keywords and expected staying time for the i-th POI, respectively.

For simplicity, we use $R(s, d)$=$\langle s, o_{x_1}, \ldots, o_{x_\psi}, d \rangle$ to represent any route from start location s to destination d that sequentially covers ψ required POIs with the ψ sets of query keywords, i.e., $\mathcal{K}_i \subseteq o_{x_i}.\mathcal{K}$. All these routes are recorded as $\mathcal{R}(s, d)$. We then have the definition of PTR query.

Definition 7.4 (Probabilistic Time-constrained Route (PTR) Query)**.** *Given an uncertain road network $\mathcal{G}$, the objective of a PTR query q is to retrieve a subset, $\mathcal{R}^h(s, d)$, of routes from $\mathcal{R}(s, d)$, i.e.,*

$$\mathcal{R}^h(s, d) = \{R | R \in \mathcal{R}(s, d) \wedge Pr_h\{R\} \geqslant \tau\} \tag{7.7}$$

where

$$Pr_h\{R\} = \left\{ \sum_{\forall w(\mathcal{G}), R \in \mathcal{R}_w^h(s,d)} Pr\{w(\mathcal{G})\} \right\} \tag{7.8}$$

and $\mathcal{R}_w^h(s, d)$ are the top-h optimal routes in possible world $w(\mathcal{G})$ such that for $\forall R \in \mathcal{R}_w^h(s, d)$

$$\mathcal{K}_i \subseteq o_{x_i}.\mathcal{K}, i = 1, 2, \ldots, \psi \tag{7.9}$$

$$[a(o_{x_i}), a(o_{x_i}) + t_i] \subseteq o_{x_i}.I \tag{7.10}$$

$$\forall R' \in \mathcal{R}(s, d) \backslash \mathcal{R}_w^h(s, d), t_w(R) \leqslant t_w(R') \tag{7.11}$$

where $t_w(R)$ is the travel time of route R in possible world $w(\mathcal{G})$.

In Definition 7.4, $Pr_h\{R\}$ is the probability that route R is among the top-h optimal routes in all possible worlds. Therefore, PTR query retrieves those routes, $\mathcal{R}^h(s,d)$, that are the top-h time-constrained routes in all possible worlds $\mathcal{W}(\mathcal{G})$ with a probability not less than threshold τ. In other words, the returned PTR routes satisfy keyword/time constraints on POIs, and have the top-h smallest travel times in high confidence.

Example 7.4. *Figure 7.1 shows an example of PTR query q, where s=v_4, d=v_5, t_s=12:00, $q.\mathcal{K}$=$\{\langle$"k_1, k_2", $30min\rangle$, $\langle$"k_6", $20min\rangle\}$, h=1, and τ=0.5. Since there are two POIs (o_2 and o_5) containing query keywords "k_1, k_2" and two POIs (o_4 and o_6) containing query keyword "k_6", we have four candidate routes, i.e., R_1=(v_4, o_2, o_4, v_5), R_2=(v_4, o_2, o_6, v_5), R_3=(v_4, o_5, o_4, v_5), and R_4=(v_4, o_5, o_6, v_5). Starting from s=v_4, the arrival time at o_5 is*

$$12:00 + \boldsymbol{t}(e_{4,5}) \cdot \frac{dist(o_5, v_4)}{|e_{4,5}|} = 12:00 + 8 \cdot \frac{1.0}{4} = 12:02$$

which is out of the time intervals of o_5 ($o_5.I$=$\{$08:00-12:00, 13:00-17:00$\}$). Therefore, both routes R_3 and R_4 cannot satisfy the service time constraints of o_5 and should be pruned.

Considering route R_1 in possible world $w_1(\mathcal{G})$, the arrival time at POI o_2 is

$$12:00 + \boldsymbol{t}(e_{4,1}) \cdot \frac{|e_{4,1}| - dist(o_2, v_1)}{|e_{4,1}|} = 12:00 + 4 \cdot \frac{3 - 1.5}{3} = 12:02$$

which is within the time interval of o_2 ($o_2.I$=$\{$9:00-13:00, 14:00-18:00$\}$). After staying at o_2 for 30 minutes, the arrival time at o_4 is 12:39 which satisfies the time constraint of o_4 ($o_4.I$=$\{$8:00-20:00$\}$). Finally, we have $\boldsymbol{t}(R_1)$=11 in possible world $w_1(\mathcal{G})$. Similarly, we have $\boldsymbol{t}(R_2)$=12 in $w_1(\mathcal{G})$. Therefore, the top-1 time-constrained route in possible world $w_1(\mathcal{G})$ is $\mathcal{R}^1_{w_1}(s,d) = \{R_1\}$.

The top-1 time-constrained routes, $\mathcal{R}_w^1(s,d)$, in other possible worlds can be computed in the same way and the corresponding results are listed in Table 7.1. Then, we have

$$Pr_h\{R_1\} = 0.144 + 0.336 + 0.036 + 0.084 = 0.6 > \tau$$

and

$$Pr_h\{R_2\} = 0.096 + 0.224 + 0.024 + 0.056 = 0.4 < \tau$$

where τ=0.5. Therefore, R_1 is the query result.

7.2.3 Problem Hardness

PTR query seeks to find a subset of routes $\mathcal{R}^h(s,d) \subseteq \mathcal{R}(s,d)$ such that these routes are among the top-h optimal routes in high confidence. Without loss of generality, we consider a special case of PTR query in which h=1 and the appearance probabilities of all possible worlds are equal.

First, we define the decision problem of PTR query as below.

Definition 7.5 (Decision-PTR)**.** *A Decision-PTR problem determines whether there exist k routes in $\mathcal{R}(s,d)$ such that these k routes contain the top-1 route in at least x possible worlds.*

Then, we have the following theorem.

Theorem 7.1. *Decision-PTR problem is NP-hard.*

Proof. We can prove that Decision-PTR problem is NP-hard by a reduction from partial covering problem [33] which has set-cover problem as one of its special cases. Given the global set $U=\{e_1, e_2, \cdots, e_m\}$ and a set of subsets $S=\{U_1, U_2, \cdots, U_n\}$ where $U_i \subseteq U$, a partial covering problem decides whether there exist k subsets $S' \subseteq S$ covering at lest x elements of U.

In Decision-PTR problem, we assume that $\mathcal{R}(s,d)=\{R_1, R_2, \cdots, R_n\}$ and $\mathcal{W}(\mathcal{G})=\{w_1, w_2, \cdots, w_m\}$. Now we convert a partial covering instance to an instance of the Decision-PTR problem. We regard each subset $U_i \in S$ as a route R_i and each element $e_j \in U$ as a possible world w_j. If $e_j \in U_i$, we have the travel time of route R_i in possible world w_j, $t_{w_j}(R_i)=1$, otherwise $t_{w_j}(R_i)=+\infty$. Thus, we can find k subsets S' from S to cover at least x elements of U if and only if we can find k routes that are the top-1 optimal route in at least x possible worlds, which is proved from the following two aspects.

- Assume that we have found k subsets S' from S such that S' covers at least x elements of U. Therefore, in each of the corresponding x possible worlds, the corresponding k routes must contain the top-1 optimal route considering that $t_{w_j}(R_i)=1$ if $e_j \in U_i$, otherwise $t_{w_j}(R_i)=+\infty$. Thus, we have at least x possible worlds of $\mathcal{W}(\mathcal{G})$ in which the corresponding k routes have the top-1 route.

- Assume that we find k routes such that they contain the top-1 route in at least x possible worlds. In each of these possible worlds, we must have that the corresponding element is covered by the corresponding k subsets since we can always find a route with travel time of 1, i.e., the top-1 route. Thus, there are at least x elements are covered.

Now we can conclude that PTR-Decision problem is NP-hard. □

One straightforward method is to enumerate all possible worlds $\mathcal{W}(\mathcal{G})$, obtain $\mathcal{R}_w^h(s,d)$ under each possible world $w(\mathcal{G})$, and aggregate all the obtained results to generate the PTR query result. However, this method is rather inefficient due to the exponential number of possible worlds that are computationally expensive to materialize. Therefore, in this work, we propose to process PTR query in a two-phase fashion. First, we design effective pruning strategies to filter out false alarms

without enumerating all possible worlds and produce a small set of candidate routes. Second, a refinement step based on Monte Carlo theory [66] is conducted to compute the final result.

7.3 Solution

7.3.1 Solution Overview

Figure 7.2 illustrates the flow chart of our proposed solution. In a nutshell, the solution covers the following three tasks:

- Generating the route search space according to the PTR query, POIs and uncertain road network (URN),
- Designing efficient pruning strategies to filter out infeasible routes to obtain a small set of candidate routes, and
- Conducting a refinement step to refine the candidate routes and get the final results,

where the first two tasks belong to the first phase while the third task belongs to the second phase.

The details of these tasks are elaborated in the following sections. In section 7.3.2, we first build indices for POIs and road network, respectively, to accelerate query processing. Section 7.3.3 discusses the generation of search space. In section 7.3.4, the details of three pruning strategies are presented. Section 7.3.5 describes how to compute candidate routes with the proposed pruning strategies and how to refine the obtained candidate routes to generate the final results as well. In addition, we also discuss about the updates of travel time samples in uncertain road network in Section 7.3.6.

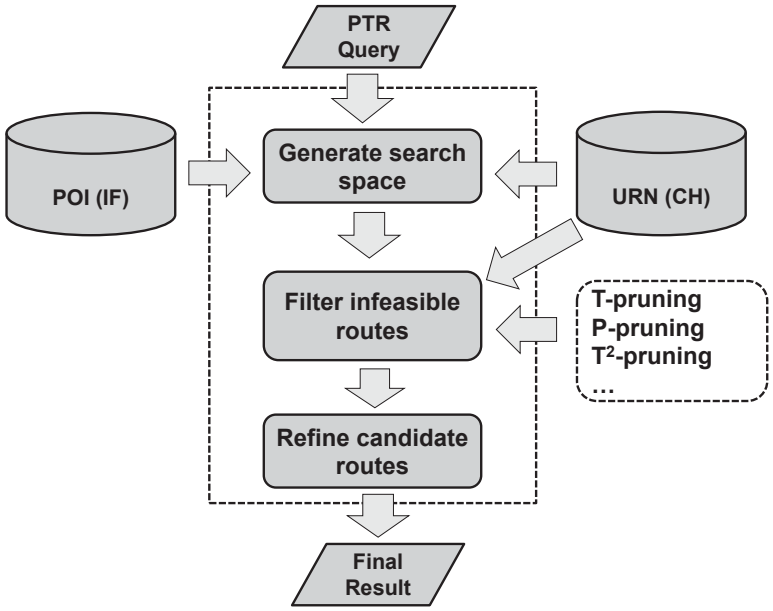


Figure 7.2: The flow chart of the proposed solution.

7.3.2 Indexing POIs and Road Networks

In order to efficiently retrieve those POIs that satisfy the query keywords during the PTR query answering, we utilize an inverted index [104] to index keywords as well as the relevant POIs. We denote $Inv(k)$ as the set of POIs containing keyword k, with cardinality $|Inv(k)|$. Given a set of query keywords $\mathcal{K}_i$=$\{k_1, \ldots, k_l\}$, the set $Inv(k_1) \cap \cdots \cap Inv(k_l)$ contains POIs that have all query keywords in $\mathcal{K}_i$. Note that, to further improve the query efficiency of accessing query keywords, we also index keyword entries in the inverted index with a B^+-tree, where the keys in the B^+-tree are unique values transformed from keywords.

In addition, in order to efficiently compute the shortest route between two POIs, we index the road networks with Contraction Hierarchy(CH) [36] which has been discussed in Section 3.2.3. Note that, CH index is adopted because we only consider the shortest distance route between each pair of POIs and the objective of PTR query is to find a sequence of required POIs with the least travel time in high confidence.

7.3.3 Generating Search Space

Given a PTR query q=(s, d, t_s, $\mathcal{K}$, h, τ), those relevant POIs are first retrieved with respect to the query keywords $q.\mathcal{K}$. For query keywords $\mathcal{K}_i \in \mathcal{K}$ (i=1, ..., ψ), all the POIs containing $\mathcal{K}_i$ are computed by using the inverted index as discussed in Section 7.3.2. If no ambiguity, we record those POIs containing all keywords in $\mathcal{K}_i$ as $Inv(\mathcal{K}_i)$ and use $|Inv(\mathcal{K}_i)|$ to represent its cardinality. Then we have the total number of combinations $\prod_{i=1}^{\psi} |Inv(\mathcal{K}_i)|$ which increases exponentially with respect to the number of required POIs. Therefore, to enumerate all these combinations will be time-consuming. Our objective is to greatly reduce the size of these combinations with novel pruning strategies as detailed in Section 7.3.4.

7.3.4 Pruning Mechanisms

In this section, we will design effective pruning methods to reduce the PTR search space, which can produce a small set of candidate routes for further refinement. Concretely, we first present *time constraint pruning* (**T-pruning**) to prune those routes that violate the time constraints of POIs. Then, *probabilistic pruning* (**P-pruning**) is proposed to enable the pruning by utilizing the probability threshold τ. Furthermore, since only top-h time-constrained routes are requested under possible worlds, we also propose the *travel time pruning* (**T^2-pruning**) that directly filters out false alarms that cannot be top-h time constrained routes.

A. Time Constraint Pruning

As illustrated in Figure 7.3, $R(s, d)$ represents any route that sequentially passes POIs o_{x_1}, o_{x_2}, ..., o_{x_ψ}, where o_{x_i} (i=1, ..., ψ) represents any POI containing query keywords $\mathcal{K}_i$. Thus, if $R(s, d)$ does not satisfy the time constraints, $o_{x_i}.I$, of some POI o_{x_i} in any possible world, $R(s, d)$ cannot be a PTR answer (i.e., $Pr_h\{R(s, d)\}$=0), and thus should be pruned.

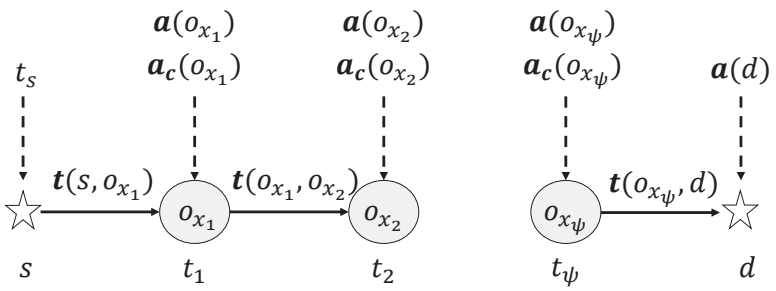


Figure 7.3: The arrival times at different POIs along route $R(s, d)$.

Specifically, let $\mathbf{a}(o_{x_i})$ be the arrival time at the i-th POI o_{x_i}, we have:

$$\mathbf{a}(o_{x_i}) = t_s + \mathbf{t}(R(s, o_{x_i})) + \sum_{j=1}^{i-1} t_j \tag{7.12}$$

where $R(s, o_{x_i})$ is the sub-route of $R(s, d)$ from s to o_{x_i}, $\mathbf{t}(R(s, o_{x_i}))$ is the travel time of $R(s, o_{x_i})$ and t_j (j=1, ..., $i-1$) is the staying time at o_{x_j}. Due to the uncertainty of $\mathbf{t}(R(s, o_{x_i}))$, $\mathbf{a}(o_{x_i})$ is uncertain and falls into the time interval $[a^-(o_{x_i}), a^+(o_{x_i})]$, where $a^-(o_{x_i})$ and $a^+(o_{x_i})$ are the earliest and latest arrival times at o_{x_i}, respectively. Then, we denote the *constrained arrival time*, $\mathbf{a}_c(o_{x_i})$, as the set of possible arrival times within the service time of o_{x_i}, i.e.,

$$\mathbf{a}_c(o_{x_i}) = \{x | x \in \mathbf{a}(o_{x_i}) \wedge [x, x + t_i] \subseteq o_{x_i}.I\} \tag{7.13}$$

Intuitively, constrained arrival time $\mathbf{a}_c(o_{x_i})$ computes arrival times such that they and their staying times are within the time intervals of o_{x_i}. Assuming that the interval of $\mathbf{a}_c(o_{x_i})$ is $[a_c^-(o_{x_i}), a_c^+(o_{x_i})]$, we have

$$[a_c^-(o_{x_i}), a_c^+(o_{x_i})] \subseteq [a^-(o_{x_i}), a^+(o_{x_i})] \tag{7.14}$$

Then, we have the following lemma about **T-Pruning**:

Lemma 7.1. *(Time Constraint Pruning, **T-Pruning**) Given an uncertain road network $\mathcal{G}$ and a PTR query q, if route $R(s, d)$ passes some POI $o_{x_i}(1 \leqslant i \leqslant \psi)$ in $\mathcal{G}$, and $|\boldsymbol{a}_c(o_{x_i})|$=0, then $R(s, d)$ can be safely pruned.*

Proof. According to the lemma assumption that $|\mathbf{a}_c(o_{x_i})|$=0, $R(s,d)$ cannot be among the top-h time-constrained routes $\mathcal{R}_w^h(s,d)$ under any possible world $w(\mathcal{G})$ due to the violation of POI time constraints. Thus, the PTR probability, $Pr_h\{R(s,d)\}$ is equal to 0, which is smaller than the nonzero probability threshold τ. Therefore, $R(s,d)$ cannot be a PTR query answer and can be pruned. □

Note that, both arrival and departure times at POIs o_{x_i} must fall into some single time interval of $o_{x_i}.I$ so that the staying time t_i can be satisfied without an interruption.

To conduct T-pruning, we need to compute $\mathbf{a}_c(o_{x_i})$ efficiently. With Eqs. (7.12) and (7.13), the arrival time and constrained arrival time at each POI can be computed iteratively as follows.

$$\mathbf{a}(o_{x_i}) = \begin{cases} t_s + \mathbf{t}(s, o_{x_i}), & i = 1 \\ \mathbf{a}_c(o_{x_{i-1}}) + t_{i-1} + \mathbf{t}(o_{x_{i-1}}, o_{x_i}), & i > 1 \end{cases} \tag{7.15}$$

where $\mathbf{t}(x,y)$ is the travel time of the shortest route from x to y. Finally, the arrival time at destination d is

$$\mathbf{a}(d) = \mathbf{a}_c(o_{x_\psi}) + t_\psi + \mathbf{t}(o_{x_\psi}, d) \tag{7.16}$$

In general, however, it is not trivial to enumerate all values of $\mathbf{a}(o_{x_i})$ and $\mathbf{a}_c(o_{x_i})$ due to their numerous instances. Instead, to avoid costly computations, we compute lower/upper bounds for $\mathbf{a}(o_{x_i})$ and $\mathbf{a}_c(o_{x_i})$.

To obtain lower/upper bounds, $a^-(o_{x_i})$ and $a^+(o_{x_i})$, of the arrival time $\mathbf{a}(o_{x_i})$ for POI o_{x_i}, we aim to obtain lower/upper bounds of $\mathbf{t}(o_{x_{i-1}}, o_{x_i})$. Denote the bounds of travel time $\mathbf{t}(x,y)$ as $[t^-(x,y), t^+(x,y)]$. Initially, for the sub-route from s to o_{x_1} on route $R(s,d)$, we have $a^-(o_{x_1})$=t_s + $t^-(s, o_{x_1})$ and $a^+(o_{x_1})$=t_s + $t^+(s, o_{x_1})$. Accordingly, the constrained arrival time interval at o_{x_1} is computed by

$$[a_c^-(o_{x_1}), a_c^+(o_{x_1})] = [a^-(o_{x_1}), a^+(o_{x_1})] \cap o_{x_1}.I', \tag{7.17}$$

where $o_{x_1}.I'$ is computed by subtracting t_1 from the upper value of each sub-interval of $o_{x_1}.I$ so that the required staying time t_1 can be satisfied. For example, if $o_{x_1}.I$=$\{$8-12, 13-16$\}$ and t_1=1, we have $o_{x_1}.I'$=$\{$8-11, 13-15$\}$.

Similarly, for the i-th POI o_{x_i} on route $R(s,d)$ $(2 \leqslant i \leqslant \psi)$, we have:

$$[a_c^-(o_{x_i}), a_c^+(o_{x_i})] = [a^-(o_{x_i}), a^+(o_{x_i})] \cap o_{x_i}.I' \tag{7.18}$$

where

$$a^-(o_{x_i}) = a_c^-(o_{x_{i-1}}) + t_{i-1} + t^-(o_{x_{i-1}}, o_{x_i}), \tag{7.19}$$

$$a^+(o_{x_i}) = a_c^+(o_{x_{i-1}}) + t_{i-1} + t^+(o_{x_{i-1}}, o_{x_i}). \tag{7.20}$$

With the bounds of constrained arrival times at POIs, route $R(s,d)$ should be pruned if there exists some POI o_{x_i} such that

$$\mathbf{a}_c(o_{x_i}) = [a^-(o_{x_i}), a^+(o_{x_i})] \cap o_{x_i}.I' = \varnothing \tag{7.21}$$

In this case, we have $|\mathbf{a}_c(o_{x_i})|$=0. Thus, with **T-pruning**, those routes that violate the time constraints of POIs can be pruned to reduce the search space.

B. Probabilistic Pruning

The basic idea of probabilistic pruning is as follows. Intuitively, if an upper bound of the PTR probability, $Pr_h\{R(s,d)\}$ for route $R(s,d)$ is smaller than the specified probability threshold τ, then $R(s,d)$ can be pruned.

Lemma 7.2. *(Probabilistic Pruning, **P-Pruning**) Denote the upper bound of the PTR probability, $Pr_h\{R(s,d)\}$, as $Pr_h^+\{R(s,d)\}$. If $Pr_h^+\{R(s,d)\}<\tau$ holds, then $R(s,d)$ can be safely pruned.*

Proof. This lemma can be easily pruned by the inequality transition of $Pr_h\{R(s,d)\}\leqslant Pr_h^+\{R(s,d)\}$ and $Pr_h^+\{R(s,d)\} < \tau$. □

Now, the task turns out to be the derivation of a probability upper bound, $Pr_h^+\{R(s,d)\}$. According to the definition of the PTR query, a PTR route should satisfy the time constraints of POIs and have the top-h smallest travel times with a probability above τ. Therefore, for route $R(s,d)$, we need to consider two factors, i.e., the time constraints of POIs and ranking probability (having the top-h smallest travel time). Note that, for route $R(s,d)$, satisfying the time constraints of POIs is the precondition of considering its ranking probability. Therefore, we will first use the constraint-based probability upper bound of $R(s,d)$ to prune this route. If $R(s,d)$ survives, its ranking probability will be further considered to compute another rank-based probability upper bound to prune this route.

(a) Constraint-based probability upper bound

According to Eq. (7.12), the arrival time at each POI o_{x_i}, $\mathbf{a}(o_{x_i})$, is a set of discrete values within $[a^-(o_{x_i}),\ a^+(o_{x_i})]$. In general, we have $[a_c^-(o_{x_i}),\ a_c^+(o_{x_i})] \subseteq [a^-(o_{x_i}),\ a^+(o_{x_i})]$, i.e., only a subset of values in $\mathbf{a}(o_{x_i})$ satisfy the time constraints of o_{x_i}. We assume that $E(o_{x_i})$ is the event that $R(s,d)$ satisfies the time constraints of o_{x_i}, i.e.,

$$Pr\{E(o_{x_i})\} = Pr\{a_c^-(o_{x_i}) \leqslant \mathbf{a}(o_{x_i}) \leqslant a_c^+(o_{x_i})\} \tag{7.22}$$

Hence, we have the constraint-based probability, $Pr_c\{R(s,d)\}$, i.e., the probability that $R(s,d)$ satisfies the time constraints at all ψ POIs, as below.

$$Pr_c\{R(s,d)\} = Pr\left\{\bigwedge_{i=1}^{\psi} E(o_{x_i})\right\}. \tag{7.23}$$

If $Pr_c\{R(s,d)\} \geqslant \tau$, we call $R(s,d)$ an *effective route*. Then, we have the following lemma.

Lemma 7.3. *Given a sub-route $R(s,o_{x_i})$ of route $R(s,d)$, we have:*

$$\begin{aligned} Pr_c\{R(s,d)\} &\leqslant Pr_c\{R(s,o_{x_i})\} \\ &\leqslant \min_{j=1,\ldots,i} Pr\{a_c^-(o_{x_j}) \leqslant \boldsymbol{a}(o_{x_j}) \leqslant a_c^+(o_{x_j})\} \end{aligned} \tag{7.24}$$

Proof. we have

$$Pr_c\{R(s, o_{x_i})\} = Pr\left\{\bigwedge_{j=1}^{i} E(o_{x_j})\right\}. \tag{7.25}$$

Given $i \leqslant \psi$, $Pr\left\{\bigwedge_{j=1}^{\psi} E(o_{x_j})\right\} \leqslant Pr\left\{\bigwedge_{j=1}^{i} E(o_{x_j})\right\}$ holds because $\bigwedge_{j=1}^{\psi} E(o_{x_j})$ has more events (i.e., more constraints) than $\bigwedge_{j=1}^{i} E(o_{x_j})$. Therefore, by combining Eqs. (7.23) and (7.25), we have $Pr_c\{R(s,d)\} \leqslant Pr_c\{P(s, o_{x_i})\}$. Furthermore, we have:

$$\begin{aligned} Pr_c\{R(s,d)\} &\leqslant Pr_c\{R(s, o_{x_i})\} \\ &= Pr\{\bigwedge_{j=1}^{i} E(o_{x_j})\} \\ &= Pr\{\bigwedge_{j=1}^{i} a_c^-(o_{x_j}) \leqslant \mathbf{a}(o_{x_j}) \leqslant a_c^+(o_{x_j})\} \\ &\leqslant \min_{j=1,\ldots,i} Pr\{a_c^-(o_{x_j}) \leqslant \mathbf{a}(o_{x_j}) \leqslant a_c^+(o_{x_j})\}. \end{aligned} \tag{7.26}$$

Hence, Lemma 7.3 holds. □

Lemma 7.3 indicates that a longer route has a lower constraint-based probability and an upper bound of the PTR probability $Pr_h\{R(s,d)\}$ can be computed by

$$\min_{j=1,\ldots,i} Pr\{a_c^-(o_{x_j}) \leqslant \mathbf{a}(o_{x_j}) \leqslant a_c^+(o_{x_j})\}. \tag{7.27}$$

To compute this upper bound, we need to calculate $Pr\{a_c^-(o_{x_j}) \leqslant \mathbf{a}(o_{x_j}) \leqslant a_c^+(o_{x_j})\}$ for POI o_{x_j}. A straightforward method is to compute the probability distribution of $\mathbf{a}(o_{x_j})$, and aggregate these samples that are within $[a_c^-(o_{x_j}), a_c^+(o_{x_j})]$ to obtain the constraint-based probability. However, this will incur a very high computation cost due to the exponential number of possible instances. Instead of computing $\mathbf{a}(o_{x_j})$ directly, we compute an upper bound for $Pr\{a_c^-(o_{x_j}) \leqslant \mathbf{a}(o_{x_j}) \leqslant a_c^+(o_{x_j})\}$ without computing the actual probability distribution.

We denote the *cumulative distribution function* (CDF) of $\mathbf{t}(R(s, o_{x_i}))$ as

$$F_R(z) = Pr\{\mathbf{t}(R(s, o_{x_i})) \leqslant z\} \tag{7.28}$$

whose lower and upper bounds are accordingly assumed to be $F_R^-(z)$ and $F_R^+(z)$, respectively. Then, we have:

$$Pr\{a_c^-(o_{x_j}) \leqslant \mathbf{a}(o_{x_j}) \leqslant a_c^+(o_{x_j})\} \leqslant F_R^+(b) - F_R^-(a) \tag{7.29}$$

where $a{=}a_c^-(o_{x_j}) - t_s - \sum_{l=1}^{j-1} t_l$, $b{=}a_c^+(o_{x_j}) - t_s - \sum_{l=1}^{j-1} t_l$, and t_l is the staying time at POI o_l. Eq. (7.29) can be proved by the following derivation:

$$\begin{aligned} Pr\{a_c^-(o_{x_j}) \leqslant \mathbf{a}(o_{x_j}) \leqslant a_c^+(o_{x_j})\} &= Pr\{a \leqslant \mathbf{a}(o_{x_j}) - t_s - \sum_{l=1}^{j-1} t_l \leqslant b\} \\ &= Pr\{a \leqslant \mathbf{t}(R(s, o_{x_j})) \leqslant b\} \\ &\leqslant F_R^+(b) - F_R^-(a) \end{aligned} \tag{7.30}$$

Next, we discuss how to compute the bounds of $F_R(z)$, i.e., $F_R^-(z)$ and $F_R^+(z)$. Assume that the CDF of $\mathbf{t}(e)$ is $F_e(x){=}Pr\{\mathbf{t}(e) \leqslant x\}$. Generally, $F_e(x)$ can be computed easily, since the distribution of $\mathbf{t}(e)$ is known to be represented by samples. However, $F_R(z)$ is difficult to compute due to the unknown distribution of $\mathbf{t}(R(s, o_{x_i}))$. Therefore, our basic idea is to generate bounds of $F_R(z)$ based on $F_e(x)$ instead of the detailed distribution of $\mathbf{t}(R(s, o_{x_i}))$. To this end, we set a certain threshold for $\mathbf{t}(e)$ ($e \in R(s, o_{x_i})$) according to z and $\mathbf{t}(e)$ such that the sum of these thresholds is less than z. Further, these thresholds are used to compute the corresponding $F_e(x)$ and $F_R(z)$ as discussed below.

We assume $R(s, o_{x_i}){=}e_1 \rightarrow e_2 \cdots \rightarrow e_l$ and specify each $e \in R(s, o_{x_i})$ a threshold:

$$t_{\triangle}(e) = t^-(e) + \frac{(t^+(e) - t^-(e)) \cdot (z - \sum_{j=1}^{l} t^-(e_j))}{\sum_{j=1}^{l} (t^+(e_j) - t^-(e_j))} \tag{7.31}$$

Then, we have the following lemma to compute bounds of $F_R(z)$:

Lemma 7.4. *Given $t_{\triangle}(e_j)$ (j=1, ..., l) in Eq. (7.31), we have three cases:*

- *Case 1: If $\sum_{j=1}^{l} t^{-}(e_j) > z$, we have $F_R(z) = 0$.*
- *Case 2: If $\sum_{j=1}^{l} t^{+}(e_j) < z$, we have $F_R(z) = 1$.*
- *Case 3: If $\sum_{j=1}^{l} t^{-}(e_j) \leqslant z \leqslant \sum_{j=1}^{l} t^{+}(e_j)$, we have*

$$\prod_{j=1}^{l} F_{e_j}(t_{\triangle}(e_j)) \leqslant F_R(z) \leqslant 1 - \prod_{j=1}^{l} (1 - F_{e_j}(t_{\triangle}(e_j))) \tag{7.32}$$

Proof. According to the definition of $\mathbf{t}(R)$, we have

$$\sum_{j=1}^{l} t^{-}(e_j) \leqslant \mathbf{t}(R) \leqslant \sum_{j=1}^{l} t^{+}(e_j), \tag{7.33}$$

thus, both Cases 1 and 2 hold. As for Case 3, we first prove the lower bound of $F_R(z)$ as follows.

$$\begin{aligned} F_R(z) &= Pr\{\mathbf{t}(s, o_{x_i}) \leqslant z\} \qquad (7.34) \\ &= \sum_{y_1+\cdots+y_l \leqslant z} Pr\{\mathbf{t}(e_1) = y_1 \wedge \cdots \wedge \mathbf{t}(e_l) = y_l\} \\ &\geqslant \sum_{y_1+\cdots+y_l \leqslant z \wedge y_j \leqslant t_{\triangle}(e_j)} Pr\{\mathbf{t}(e_1) = y_1 \wedge \cdots \wedge \mathbf{t}(e_l) = y_l\} \end{aligned}$$

According to the definition of $t_{\triangle}(e_i)$, if $y_i \leqslant t_{\triangle}(e_i)$ holds for $i = 1, \ldots, l$, we have

$\sum_{i=1}^{l} x_i \leqslant \sum_{i=1}^{l} t_{\triangle}(e_i)$ and further:

$$\begin{aligned}
\sum_{i=1}^{l} t_{\triangle}(e_i) &= \sum_{i=1}^{l} (t^{-}(e_i) + \frac{(t^{+}(e_i) - t^{-}(e_i)) \cdot (z - \sum_{j=1}^{l} t^{-}(e_j))}{\sum_{j=1}^{l} (t^{+}(e_j) - t^{-}(e_j))}) \\
&= \sum_{i=1}^{l} t^{-}(e_i) + \frac{(z - \sum_{j=1}^{l} t^{-}(e_j)) \cdot \sum_{i=1}^{l} (t^{+}(e_i) - t^{-}(e_i))}{\sum_{j=1}^{l} (t^{+}(e_j) - t^{-}(e_j))} \\
&= \sum_{i=1}^{l} t^{-}(e_i) + z - \sum_{j=1}^{l} t^{-}(e_j) \\
&= z
\end{aligned}$$

Therefore, inequality $\sum_{i=1}^{l} x_i \leqslant z$ always holds when $y_i \leqslant t_{\triangle}(e_i) (1 \leqslant i \leqslant l)$. Moreover, according to Eq. (7.34), we have:

$$\begin{aligned}
F_R(z) &\geqslant \sum_{y_1 + \cdots + y_l \leqslant z \wedge y_j \leqslant t_{\triangle}(e_j)} Pr\{\mathbf{t}(e_1) = y_1 \wedge \cdots \wedge \mathbf{t}(e_l) = y_l\} \\
&= Pr\{\mathbf{t}(e_1) \leqslant t_{\triangle}(e_1) \wedge \cdots \wedge Pr(\mathbf{t}(e_l) \leqslant t_{\triangle}(e_l)\} \\
&= \prod_{j=1}^{l} Pr\{\mathbf{t}(e_j) \leqslant t_{\triangle}(e_j)\} = \prod_{j=1}^{l} F_{e_j}(t_{\triangle}(e_j))
\end{aligned}$$

the left hand side of Eq. (11) holds.

To prove the right hand side of Eq. (11), we need to prove that $1 - F_R(z) \geqslant \prod_{i=1}^{l} (1 - F_{e_i}(t_{\triangle}(e_i)))$, i.e.,

$$Pr\{\mathbf{t}(R(s, o_{x_i})) \geqslant z\} \geqslant \prod_{i=1}^{l} (1 - F_{e_i}(t_{\triangle}(e_i))). \tag{7.35}$$

Similarly, we have

$$\begin{aligned} Pr\{\mathbf{t}(s, o_{x_i}) \geqslant z\} &= \sum_{y_1+\cdots+y_l \geqslant z} Pr\{\mathbf{t}(e_1) = y_1 \wedge \cdots \wedge \mathbf{t}(e_l) = y_l\} \\ &\geqslant \sum_{y_1+\cdots+y_l \geqslant z \wedge y_j \geqslant t_{\triangle}(e_j)} Pr\{\mathbf{t}(e_1) = y_1 \wedge \cdots \wedge \mathbf{t}(e_l) = y_l\} \\ &= Pr\{\mathbf{t}(e_1) \geqslant t_{\triangle}(e_1), \ldots, Pr(\mathbf{t}(e_l) \geqslant t_{\triangle}(e_l)\} \\ &= \prod_{i=1}^{l}(1 - F_{e_i}(t_{\triangle}(e_i))), \end{aligned}$$

and the right hand side of Eq. (11) also holds. □

From Lemma 7.4, we obtain bounds for $F_R(z)$, i.e., in Cases 1 and 2, we have $F_R^-(z)$=$F_R^+(z)$=0 (or 1); in Case 3, we have

$$F_R^-(z) = \prod_{i=1}^{l} F_{e_i}(t_{\triangle}(e_i)) \tag{7.36}$$

and

$$F_R^+(z) = 1 - \prod_{i=1}^{l}(1 - F_{e_i}(t_{\triangle}(e_i))) \tag{7.37}$$

Further, since it holds that $Pr\{a \leqslant \mathbf{t}(R(s, o_{x_i})) \leqslant b\} = F_R(b) - F_R(a)$, we can compute bounds for $Pr\{a \leqslant \mathbf{t}(R(s, o_{x_i})) \leqslant b\}$, i.e.,

$$Pr\{a \leqslant \mathbf{t}(R(s, o_{x_i})) \leqslant b\} \geqslant F_R^-(b) - F_R^+(a) \tag{7.38}$$

and

$$Pr\{a \leqslant \mathbf{t}(R(s, o_{x_i})) \leqslant b\} \leqslant F_R^+(b) - F_R^-(a) \tag{7.39}$$

Then, with Eq. (7.29), route $R(s, d)$ can be pruned if there exists a sub-route $R(s, o_{x_j})$ (j=1, …, i) such that $F_R^+(b) - F_R^-(a) \leqslant \tau$.

(b) Rank-based probability upper bound

The constraint-based upper bound only considers the time constraints of POIs. In this subsection, we will derive a rank-based probability upper bound for filtering out false alarms further.

Assume that we have obtained a set of effective routes (as discussed in Section 7.3.4), U=$\{R_1, \cdots, R_n\}$, sorted by $t^-(\cdot)$ where $Pr_c\{R_i\}\geqslant\tau(1\leqslant i\leqslant n)$. We record U_i=$\{R_1, \cdots, R_i\}(i \leqslant n)$. For route R_i, if there are at least h routes in U_{i-1} having travel time less than R_i, then R_i has a rank larger than h. However, a strict ranking among these routes cannot be implemented because the travel time of each route is a random variable bounded by an interval. Our idea is to compute the probability that there are at least h routes in U_{i-1} having travel time less than R_i, i.e.,

$$\sum_{A\subseteq U_{i-1}\wedge|A|\geqslant h} Pr\{\bigwedge_{R\in A}\mathbf{t}(R)\leqslant\mathbf{t}(R_i)\}\cdot Pr\{\bigwedge_{R\in U_{i-1}\setminus A}\mathbf{t}(R)>\mathbf{t}(R_i)\} \tag{7.40}$$

Generally, it is not trivial to enumerate all such subsets to compute the exact probability. Hence, we only consider the first h routes and have the following lemma.

Lemma 7.5. *Given a set of constraint-based routes U=$\{R_1, \cdots, R_n\}$ sorted by $t^-(.)$, for route $R_i(i > h)$, if $1-\prod_{R\in U_h} LB\_F_R(t^-(R_i))<\tau$, then routes in $U\setminus U_{i-1}$ can be pruned.*

Proof. First, we have

$$\begin{aligned}
\sum_{A\subseteq U_{i-1}\wedge|A|\geqslant h} Pr\{\bigwedge_{R\in A}\mathbf{t}(R)\leqslant\mathbf{t}(R_i)\}\cdot Pr\{\bigwedge_{R\in U_{i-1}\setminus A}\mathbf{t}(R)>\mathbf{t}(R_i)\} &\geqslant Pr\{\bigwedge_{R\in U_h}\mathbf{t}(R)\leqslant\mathbf{t}(R_i)\}\\
&\geqslant\prod_{R\in U_h}Pr\{\mathbf{t}(R)\leqslant\mathbf{t}(R_i)\}\\
&\geqslant\prod_{R\in U_h}Pr\{\mathbf{t}(R)\leqslant t^-(R_i)\}\\
&\geqslant\prod_{R\in U_h}F_R^-(t^-(R_i))
\end{aligned}$$

Therefore, we further have

$$Pr_h\{R_i\} \leqslant 1 - \prod_{R \in U_h} F_R^-(t^-(R_i)) \tag{7.41}$$

If $1 - \prod_{R \in U_h} F_R^-(t^-(R_i)) < \tau$, by the inequality transition, we have $Pr_h\{R_i\} < \tau$, thus R_i can be safely pruned. Meanwhile, for each route R_t in $U \backslash U_i = \{R_{i+1}, \cdots, R_n\}$, we have

$$Pr_h\{R_t\} \leqslant 1 - \prod_{R \in U_h} F_R^-(t^-(R_t)) \leqslant 1 - \prod_{R \in U_h} F_R^-(t^-(R_i)) < \tau \tag{7.42}$$

Therefore, $U \backslash U_{i-1}$ can be pruned if $1 - \prod_{R \in U_h} F_R^-(t^-(R_i)) < \tau$. □

Thus, a candidate route that cannot be pruned by the constraint-based probability upper bound might be filtered out by using this rank-based bound. In addition, a sub-route can also be pruned by using this bound as described below.

Lemma 7.6. *Given a set of constrained routes* $U = \{R_1, \ldots, R_n\}$ *sorted by* $t^-(\cdot)$ *and a sub-route* $R(s, o_{x_i})$, $R(s, o_{x_i})$ *can be pruned if*

$$1 - \prod_{R \in U_h} F_R^-(t^-(R(s, o_{x_i}))) < \tau \tag{7.43}$$

Proof. A complete route generated from $R(s, o_{x_i})$ has a larger $t^-(\cdot)$ value than that of $R(s, o_{x_i})$, which reduces its corresponding PTR probability, hence the proof. □

Thus, $1 - \prod_{R \in U_h} F_R^-(t^-(R))$ can be used as the rank-based probability upper bound to prune candidate routes.

(c) Discussion on two probability upper bounds

During the query processing, both constraint-based and rank-based upper bounds of the PTR probability are employed to prune routes. The constraint-based probability upper bound can prune those routes that have constraint-based probabilities less

than τ, which mainly considers the keyword/time constraints of POIs. For the rank-based probability upper bound, it requires that there exist h candidate routes whose constraint-based probabilities are above τ. Thus, we can also use it to prune those candidate routes whose PTR probabilities less than τ. Therefore, the rank-based probability works as a complement to the constraint-based one to collaboratively reduce the number of candidate routes.

C. Travel Time Pruning

According to the definition of the PTR query, we aim to compute the probabilities that routes are among the top-h time-constrained routes which satisfy both keyword and time constraints and have the top-h smallest travel times. With this observation, we further proposed the *travel time pruning*, denoted as **T^2-Pruning**. The basic idea of **T^2-Pruning** is to use a travel time threshold to directly prune those routes with larger travel times than h constraint-based routes we have seen so far without computing probability upper bounds. Concretely, we have the **T^2-Pruning** in the following lemma.

Lemma 7.7. *(Travel Time Pruning, **T^2-Pruning**) Assume that we have obtained h effective routes which satisfy time constraints at POIs on these routes with probability above τ. Let ρ be the largest travel time upper bound among these h routes. Then, a route $R(s,d)$ can be pruned if its lower bound of travel time, $t^-(R(s,d))$, is larger than ρ.*

Proof. From the lemma assumption, if $t^-(R(s,d)) > \rho$ holds, $R(s,d)$ has larger travel time than at least h routes in all possible worlds (since we have obtained h routes that satisfy keyword/time constraints at POIs). Hence, route $R(s,d)$ cannot be a top-h time-constrained route and should be pruned. □

We consider two cases of obtaining h effective routes used in Lemma 7.7. First,

for $R(s,d)$, if the arrival time at every POI on $R(s,d)$ is within the corresponding service time, i.e.,

$$[a_c^-(o_{x_i}), a_c^+(o_{x_i})] = [a^-(o_{x_i}), a^+(o_{x_i})] \tag{7.44}$$

then route $R(s,d)$ is an effective route. The second case is that, there exist some POIs o_{x_i} on $R(s,d)$ whose time intervals do not fully cover the whole arrival time, i.e.,

$$[a_c^-(o_{x_i}), a_c^+(o_{x_i})] \neq [a^-(o_{x_i}), a^+(o_{x_i})] \tag{7.45}$$

In this case, our rationale is to derive an lower bound, $Pr_c^-\{R(s,d)\}$, for the probability $Pr_c\{R(s,d)\}$. If this lower bound is above threshold τ, then route $R(s,d)$ is an effective route. According to the definition of $Pr_c\{R(s,d)\}$, we have

$$\begin{aligned} Pr_c\{R(s,d) &\geqslant Pr_c\{R(s,o_{x_1})\} \cdot \prod_{i=2}^{\psi} Pr_c\{R(o_{x_{i-1}}, o_{x_i})\} \\ &\geqslant Pr_c^-\{R(s,o_{x_1})\} \cdot \prod_{i=2}^{\psi} Pr_c^-\{R(o_{x_{i-1}}, o_{x_i})\}. \\ &\equiv Pr_c^-\{R(s,d)\} \end{aligned} \tag{7.46}$$

where $Pr_c^-\{R(x,y)\}$ is a lower bound for $Pr_c\{R(x,\, y)\}$.

Now the goal turns out to compute the lower bounds of $Pr_c\{R(s,o_{x_1})\}$ and $Pr_c\{R(o_{x_{i-1}}, o_{x_i})\}$ $(2 \leqslant i \leqslant \psi)$. Obviously, if there exists a travel time interval for $\mathbf{t}(R(x,y))$ in which $R(x,y)$ always satisfies the keyword/time constraints of POIs, we can compute a lower bound for the probability that the travel time $\mathbf{t}(x,y)$ falls into this interval by using Eq. (7.38). Based on the constrained arrival time at $o_{x_{i-1}}$ and o_{x_i}, the constraint-based interval of $\mathbf{t}(R(o_{x_{i-1}}, o_{x_i}))$, i.e.,

$$[t_c^-(R(o_{x_{i-1}}, o_{x_i})), t_c^-(R(o_{x_{i-1}}, o_{x_i}))] \tag{7.47}$$

can be computed by solving the following inequalities:

$$\begin{cases} t^-(R(o_{x_{i-1}}, o_{x_i})) \leqslant x \leqslant t^+(R(o_{i-1}, o_{x_i})), \\ a_c^-(o_{x_i}) \leqslant a_c^-(o_{x_{i-1}}) + x + t_{i-1} \leqslant a_c^+(o_{x_i}) \\ a_c^-(o_{x_i}) \leqslant a_c^+(o_{x_{i-1}}) + x + t_{i-1} \leqslant a_c^+(o_{x_i}). \end{cases} \tag{7.48}$$

Then, with Eq. (7.38), we can calculate a lower bound of $Pr_c\{R(o_{x_{i-1}}, o_{o_{x_i}})\}$ below:

$$\begin{aligned} Pr_c\{R(o_{i-1}, o_{x_i})\} &\geqslant Pr\{t_c^-(R(o_{x_{i-1}}, o_{x_i})) \leqslant \mathbf{t}(R(o_{x_i}, o_{x_{i+1}})) \leqslant t_c^+(R(o_{i-1}, o_i))\} \\ &\geqslant F^-(t_c^+(R(o_{i-1}, o_{x_i}))) - F^+(t_c^-(R(o_{x_{i-1}}, o_{x_i}))). \\ &\equiv Pr_c^-\{R(o_{i-1}, o_{x_i})\}. \end{aligned} \tag{7.49}$$

With $Pr_c^-\{R(s, o_{x_1})\}$ and $Pr_c^-\{R(o_{x_{i-1}}, o_{x_i})\}$, we can compute the lower bound of the constraint-based probability in Eq. (7.46). If this constraint-based probability is above τ, then we can use $R(s, d)$ as an effective route to filter out other false alarms as described in Lemma 7.7.

7.3.5 Query Processing Algorithm

To efficiently answer PTR queries, we first generate a small set of candidate routes by utilizing the designed pruning strategies. After that, a refinement step based on sampling is conducted to compute the final query results.

A. Candidate Route Generation

We propose an expansion-based heuristic algorithm which has the similar flavor to A* algorithm. We expand from POI by POI and the shortest route between two POIs is computed by using the CH index. During the expansion, **T-pruning**, **P-pruning** and **T^2-pruning** are employed together to prune infeasible routes.

The PTR query processing algorithm is elaborated in Algorithm 7.1 which accepts as input a PTR query q and produces as output a small set of candidate routes C. Initially, three priority queues C, Q and U are created to store candidate routes,

Algorithm 7.1 Generate Candidate Routes

Input: a PTR query q=$(s, d, t_s, \mathcal{K}, h, \tau)$
Output: Candidate routes set C

```
1: Initialize C, Q, U ←new priorityQueue()
2: ρ ← +∞
3: R ← ⟨s⟩
4: Q.enqueue(R,t_est(R))
5: while Q is not empty do
6:     R ← Q.dequeue()
7:     if t_est(R) ⩾ ρ then
8:         Break
9:     i ←POICount(R)
10:    o_curr ←currPOI(R)
11:    if i = ψ then
12:        R_new ← R ⊕ R(o_curr, d)
13:        if t⁻(R_new) < ρ then
14:            C.add(R_new, t⁻(R_new))
15:        if Pr_c{R_new} ⩾ τ then
16:            U.add(R_new, t⁻(R_new))
17:            P-Pruning(C)
18:            update(ρ)
19:    else
20:        R_new ← P.expand()
21:        if not (t⁻(R_new) ⩾ ρ and T-Pruning(R_new) and P-Pruning(R_new)) then
22:            Q.enqueue(R_new, t_est(R_new))
23:        update R
24:        Q.enqueue(R, t_est(R))
25: return C
```

expanded routes which have covered partial POIs, and constraint-based routes, respectively (line 1). All these routes are sorted by the lower bound of travel time. In addition, ρ records the travel time threshold of the current top-h constraint-based routes (line 2).

First, the route containing the start location s is added to Q (lines 3-4) and its weight is an estimated value of the travel time, $t_{est}(R)$. Assume that the current POI of R is o_{curr} (initially, o_{curr}=s) and the next POI (if exists) is o_{next}. Meanwhile, $Euc(x, y)$ is the Euclidean distance between x and y, and $vel^{+}(\mathcal{G})$ is the maximum velocity over the whole road network. Then, we compute $t_{est}(R)$ as below:

- Case 1: R has covered all required POIs.

$$t_{est}(R) = t^{-}(R) + \frac{Euc(o_{curr}, d)}{vel^{+}(\mathcal{G})} \tag{7.50}$$

- Case 2: R only covers partial POIs.

$$t_{est}(R) = t^{-}(R) + \frac{Euc(o_{curr}, o_{next}) + Euc(o_{next}, d)}{vel^{+}(\mathcal{G})} \tag{7.51}$$

In each loop, while Q is not empty (line 5), the route R with the minimum $t_{est}(R)$ is dequeued from Q (line 6). If $t_{est}(R) \geqslant \rho$ (line 7), algorithm terminates, otherwise we have two cases as described below:

- **Case 1**: R has covered all the required POIs (line 11). In this case, we expand from R to compute the shortest route to d and generate a complete route, R_{new}, from s to d. If R_{new} survives from T^2-pruning, it is returned as a candidate route (line 14). Additionally, if $Pr_c\{R_{new}\} \geqslant \tau$, R_{new} is added to U (line 16); each route in C is checked by P-pruning (line 17); and ρ is updated (line 18).

- **Case 2**: R only covers partial POIs (line 19). In this case, we expand from R to cover the $(i+1)$-th POI, where i is the number of POIs that R has already covered, and a new route, R_{new}, is generated. Then, three pruning methods are employed. If R_{new} survives, we add it to Q with the estimated travel time. Meanwhile, R is updated (the next POI, o_{next}) and inserted into Q again.

The loops terminate when the $t_{est}(\cdot)$ value of dequeued route is larger than ρ, or Q is empty. Finally, a set C of candidate routes is returned.

B. Refinement

Generally, the number of returned candidate routes is small but some of them may be not PTR answers. Hence, a refinement step is required to compute the real results. In order to efficiently obtain the final result, we use a sampling algorithm, as

described in Algorithm 7.2, to estimate the probabilities of being among the top-h time-constrained routes for candidate routes. Particularly, in line 4, function *sample*($\mathcal{G}$) samples the travel time of each edge according to its probability distribution. Here, only the edges covered by the routes in U are sampled, since other edges of $\mathcal{G}$ have no effect on the final result. According to the Monte Carlo theory [66], we have:

$$Pr\{|Pr_h(R) - \hat{Pr}(R)| \leqslant \eta\} > 1 - \xi \tag{7.52}$$

where $\hat{Pr}(R)$=N_0/N is computed according to Algorithm 7.2, and both parameters ξ and η are set to 0.1 by default [66] (line 2). Finally, those routes with N_0/N larger than τ are returned as the final query result.

Algorithm 7.2 Calculate $\hat{Pr}(R)$

Input: $R \in C$
Output: $\hat{Pr}(R)$
1: $N_0 \leftarrow 0$
2: $N \leftarrow \frac{4(|C|-1)\ln(2/\xi)}{\eta}$
3: **for** $i \leftarrow 1$ to N **do**
4: $w_i \leftarrow$ sample($\mathcal{G}$)
5: compute $\mathcal{R}^h_{w_i}(s,d)$ among U
6: **if** $P \in \mathcal{R}^h_{w_i}(s,d)$ **then**
7: $N_0 \leftarrow N_0$+1
8: **return** N_0/N

7.3.6 Updates of URNs

In order to capture the dynamic uncertainty of the travel times on roads over uncertain road network $\mathcal{G}$, we maintain a queue for each edge $e_{i,j}$. The queue contains T samples, s_r ($t_0 - T + 1 \leqslant r \leqslant t_0$), for the last T timestamps within $[t_0 - T + 1, t_0]$, where t_0 is the current timestamp. Each sample is represented by a triple $(e_{i,j}, s_r, r)$, where $e_{i,j}$ is an edge, s_r is a possible travel time value on edge $e_{i,j}$, and r is the timestamp that the sample is collected.

In order to facilitate PTR query processing, we maintain some pre-computed data for the URN $\mathcal{G}$, including the minimum/maximum travel time, $t^-(e_{i,j})$ and $t^+(e_{i,j})$, on each edge $e_{i,j}$, and the minimum/maximum velocities, denoted as $vel^-(\mathcal{G})$ and $vel^+(\mathcal{G})$, on the entire road network $\mathcal{G}$. At a new timestamp $(t_0 + 1)$, a new sample, $s_{(t_0+1)}$, of the travel time on edge $e_{i,j}$ arrives and is added to the queue. Meanwhile, the oldest one, $s_{(t_0-T+1)}$, is expired and removed from the queue, and the pre-computed data is updated as below.

- **Update $t^-(e_{i,j})$ and $t^+(e_{i,j})$**: If the new sample $s_{(t_0+1)}$ is within interval $[t^-(e_{i,j}), t^+(e_{i,j})]$, we do not need to update the time interval. Otherwise, if $s_{(t_0+1)}<t^-(e_{i,j})$, we update $t^-(e_{i,j})$ with $s_{(t_0+1)}$; if $s_{(t_0+1)}>t^+(e_{i,j})$, we update $t^+(e_{i,j})$ with $s_{(t_0+1)}$. If the expired sample $s_{(t_0-T+1)} \in [t^-(e_{i,j}), t^+(e_{i,j})]$, we do not need to update the time interval. If $s_{(t_0-T+1)}$ equals to either $t^-(e_{i,j})$ or $t^+(e_{i,j})$, we need to scan all the samples in the last T timestamps to re-calculate the interval $[t^-(e_{i,j}), t^+(e_{i,j})]$.

- **Update $vel^-(\mathcal{G})$ and $vel^+(\mathcal{G})$**: If the travel time interval $[t^-(e_{i,j}), t^+(e_{i,j})]$ remains the same, we do not update $vel^-(\mathcal{G})$ and $vel^+(\mathcal{G})$. Otherwise, if the minimum travel time $t^-(e_{i,j})$ has changed, we have the new maximum velocity on edge $e_{i,j}$, i.e., $vel^+_{new}(e_{i,j})=|(e_{i,j}|/t^-(e_{i,j})$, and update $vel^+(\mathcal{G})$ with $vel^+_{new}(e_{i,j})$ when $vel^+_{new}(e_{i,j})>vel^+(\mathcal{G})$, or re-compute $vel^+(\mathcal{G})$ when the old maximum velocity $vel^+(e_{i,j})=vel^+(\mathcal{G})$ and $vel^+(e_{i,j})>vel^+_{new}(e_{i,j})$. If the maximum travel time $t^+(e_{i,j})$ has changed, we have the new minimum velocity on edge $e_{i,j}$, i.e., $vel^-_{new}(e_{i,j})=|e_{i,j}|/t^+(e_{i,j})$, and update $vel^-(\mathcal{G})$ with $vel^-_{new}(e_{i,j})$ when $vel^-_{new}(e_{i,j})<vel^-(\mathcal{G})$ or re-compute $vel^-(\mathcal{G})$ when the old minimum velocity $vel^-(e_{i,j})=vel^-(\mathcal{G})$ and $vel^-(e_{i,j})<vel^-_{new}(e_{i,j})$.

7.4 Experiments

7.4.1 Setups

In this section, we conduct extensive experiments on three real datasets (i.e., California, Beijing and London) to evaluate the performance of proposed solution. Specifically, California dataset (including road network and POIs) is from *Real Datasets for Spatial Databases*[1]; Beijing dataset has road network from *OpenStreetMap*[2] (OSM) and POIs from *Datatang*[3]; London dataset extracts both road network and POIs from OSM. The statistical information of three datasets, including index construction time and index size for both Inverted Index (Inv) and Contraction Hierarchy (CH), is detailed in Table 7.3.

Table 7.3: Datasets for evaluating PTR query

Attributes	**California**	**Beijing**	**London**
number of vertices	21,048	46,029	209,045
number of edges	21,693	62,778	282,267
number of POIs	104,470	252,125	34,341
Index size (Inv) (MB)	0.5	1.2	0.3
Index construction time (Inv) (sec)	0.11	0.595	0.043
Index size (CH) (MB)	44	152	341
Index construction time (CH) (sec)	2.2	28.7	47.9

Table 7.4: Parameter setting in PTR query

Parameters	**Values**
$\|q.\mathcal{K}\|$	1,**2**,3,4,5
h	1, **3**, 5, 7, 9
τ	0.1, 0.3, **0.5**, 0.7, 0.9

Keywords of POIs: The POIs of California have 62 categories and that of Beijing have 533 categories. In the experiments, each category is regarded as the keyword description of POIs, i.e., $o_i.\mathcal{K}$, in this category. Each POI in London is associated

[1] http://www.cs.utah.edu/ lifeifei/SpatialDataset.htm

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

[3] http://www.datatang.com/

with a set of keywords, extracted from OSM, to describe its properties and services.

Time intervals of POIs: We define three time interval patterns, i.e., {7:00-12:00,13:00-17:00} (such as banks), {8:00-22:00} (such as restaurants), and {0:00-24:00} (such as 24-hour stores). Then, one of the three patterns is selected for each POI. After that, to increase the diversity of the time intervals of POIs, for each POI, we generate sub-intervals of the selected time interval pattern such that they have an overlap of more than 70%. For example, assuming that a time interval in the pattern is $[a, b]$, we randomly generate a sub-interval $[x, y]$ within $[a, b]$ such that $\frac{y-x}{b-a} \geqslant 0.7$.

Travel time of roads: For each road, we first randomly select a speed sub-interval within [20km/h, 80km/h] which is the driving speed interval for most cities. Then, a set, $\mathcal{V}$, of $T(=50)$ speed samples is randomly generated within the sub-interval according to ***Uniform distribution*** and ***Truncated Gaussian distribution***, respectively. In addition, we also extract ***Real*** speed samples from the GPS trajectories of over 10,000 taxis in Beijing dataset[4]. Then, with the length of each road e, each sample of $\mathbf{t}(e)$ is obtained by $\frac{|e|}{v}$ ($v \in \mathcal{V}$). Note that, truncated gaussian distribution is used as the default distribution.

Queries: For each dataset, we evaluate 50 PTR queries whose start location and destination are generated randomly within the road network area and their departure times are randomly selected between 6:00 and 22:00. Additionally, the query keywords are also randomly selected from the corresponding vocabulary of each dataset. The staying time at each POI is randomly selected between 600 and 1,200 seconds.

Table 7.4 illustrates experimental settings, where the default values of parameters are highlighted in bold font. All experiments are run on a PC with a 3.1GHz Intel processor and a 8GB RAM.

[4] If there are no enough samples for some roads, we add the average values of existing samples to ensure T samples.

7.4.2 Experimental Results

Pruning power vs. datasets: Figure 7.4 illustrates the relationship of candidate route sets after applying pruning strategies, i.e., **T-pruning** (T), **P-pruning** (P), and **T^2-pruning** (T^2). We can see that T-pruning results in a superset of candidate sets from the other two pruning strategies. Moreover, the intersection of two candidate route sets by applying P-pruning and T^2-pruning is our final candidate set that needs further refinement. Figure 7.5 illustrates the performance of three pruning strategies over three datasets with Gaussian vehicle speeds (the case of datasets with Uniform vehicle speeds is similar and thus omitted). As presented, a considerable portion of infeasible routes are pruned when only T-pruning is employed. Further, compared with T-pruning, the number of candidate routes after T+P pruning is greatly reduced. Finally, we can get a small set of candidate routes with all the three pruning strategies, i.e., T+P+T^2 pruning. Furthermore, we can see from Figure 7.5 that the number of final candidate routes (after all pruning) on each dataset has almost the same scale as that of final results (obtained after the verification step given in Algorithm 7.2), which confirms the effectiveness of our pruning strategies.

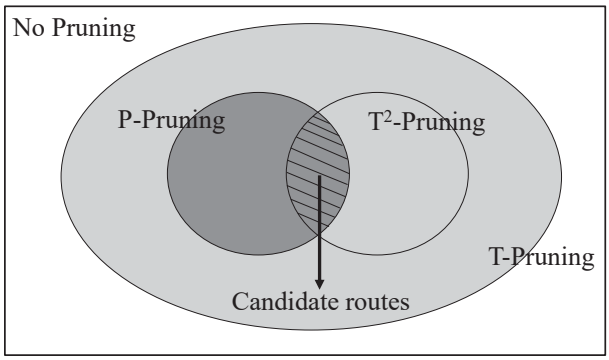


Figure 7.4: The relationship between three pruning methods

Method comparison: We compare the performance of our solution with that of the solution by enumerating all possible worlds [1] (EPW for short). As illustrated in Figure 7.6, the response time of EPW is much larger than that of our solution

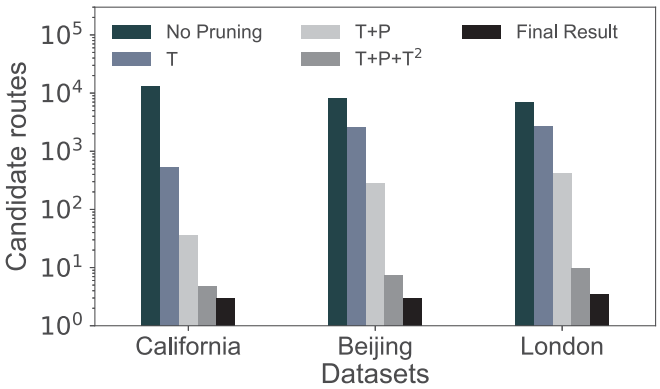


Figure 7.5: Pruning ability test on different datasets

on all three datasets. Actually, when the number of query keywords increases, it is computationally prohibitive to process the PTR query with EPW solution. On the contrary, our solution can always compute the PTR query efficiently.

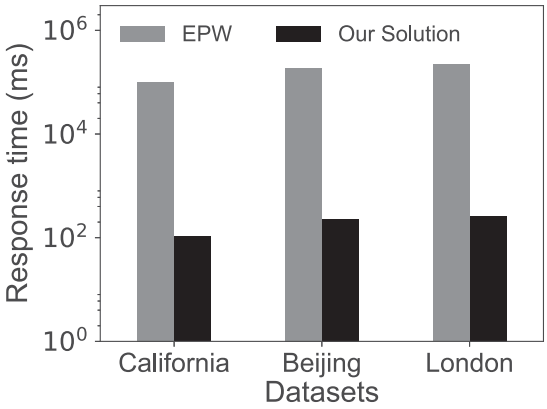
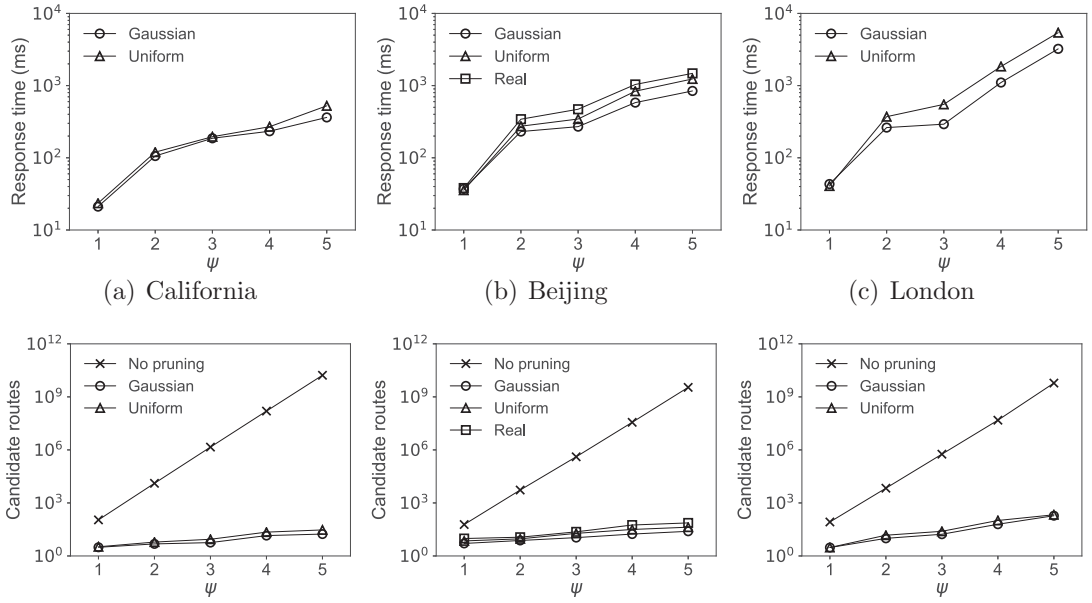


Figure 7.6: The response time comparison between EPW and our solution

Performance vs. the number of keyword sets $|\mathcal{K}|$: Figure 7.7 illustrates the experimental results while varying the number of query keyword sets, i.e., $|\mathcal{K}|$, on different datasets with Gaussian, Uniform, and Real distribution of the travel time on each edge. With the increase of $|\mathcal{K}|$, the response time increases since more routes should be evaluated. Nonetheless, with $|\mathcal{K}|$ less than 4 (the general case in reality), we can efficiently get the final results within around one second. With the increase of $|\mathcal{K}|$, the number of candidate routes increases exponentially without applying any pruning methods. After using the proposed pruning strategies, however, the number



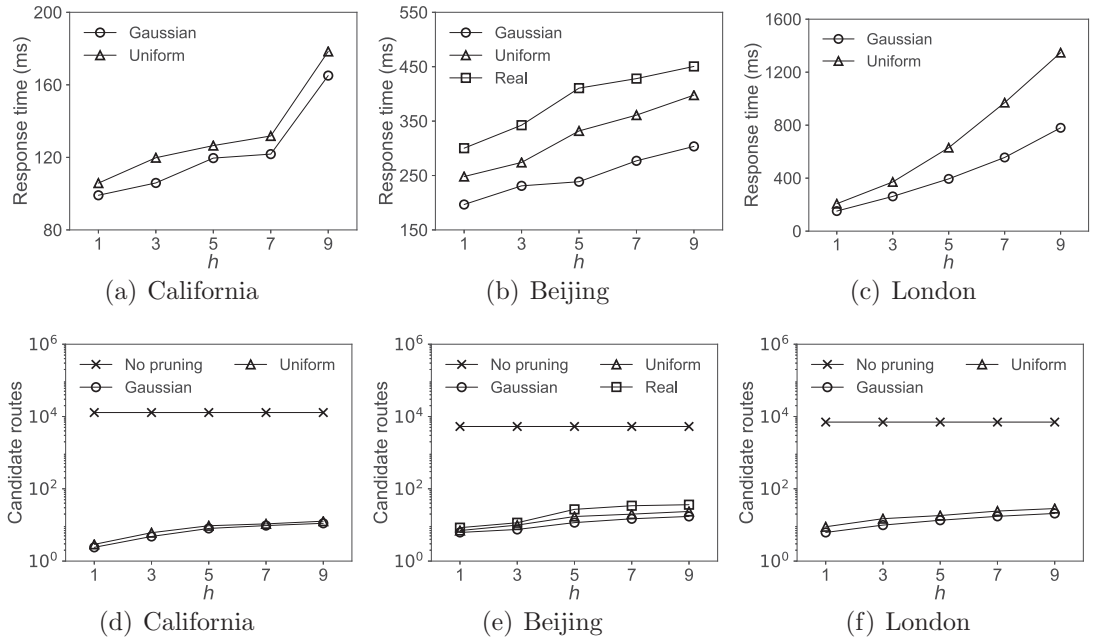
(a) California (b) Beijing (c) London

(d) California (e) Beijing (f) London

Figure 7.7: The results while varying the number of keyword sets $|q.\mathcal{K}|$

of candidate routes slightly increases, and remains small. For example, the number of candidate routes without pruning is in the scale of millions when $|\mathcal{K}|$=3, and the number of candidate routes after T+P+T^2 pruning is only in the scale of tens. Meanwhile, with Uniform distribution of $\mathbf{t}(e)$, there are more candidate routes than that of truncated Gaussian distribution, which accordingly takes higher response time. This is because Gaussian distribution concentrates around the mean value, which makes the P-pruning more effective. The real distribution takes even more time and generates more candidate routes because its speed span is larger than that of other two distribution.

Performance vs. parameter h: Figure 7.8 presents the response time and the number of candidate routes by varying h, where other parameters are set to their default values. We can see that the number of candidate routes has a smooth increase, since larger h means more routes need to be retrieved and checked. This further incurs an increase in the query response time for large h, as shown in Figure 7.8.

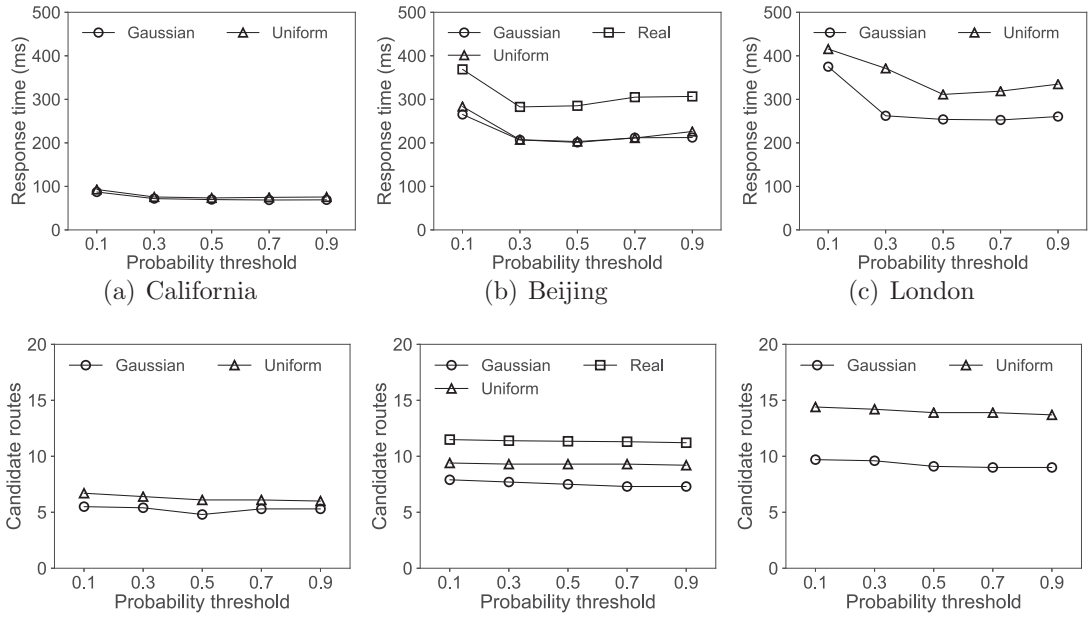


(a) California (b) Beijing (c) London

(d) California (e) Beijing (f) London

Figure 7.8: The results while varying parameter h

Performance vs. probability threshold τ: Figure 7.9 reports the experimental results with different probability thresholds from 0.1 to 0.9. According to Figure 7.9, the curves for response time are almost concave. The reason is as follows. On the one hand, a small τ covers more candidate routes but generates a good pruning threshold ρ in Algorithm 7.1. On the other hand, a large τ can prune more expanding routes during the query processing but impedes the generation of the pruning threshold ρ. Therefore, with a small τ (e.g., 0.1), a large number of evaluated routes incurs a high response time. With the increase of τ, the number of evaluated routes decreases, which results in the decrease of the response time. However, when τ becomes larger, Algorithm 7.1 will postpone to terminate because the pruning threshold ρ is difficult to generate (since fewer routes satisfy the constraint of the probability threshold τ). Thus, the response time first decreases, and then increases again for large τ. With the increase of τ, the number of candidate routes slightly decreases because fewer complete routes satisfy the probabilistic requirement.



(a) California (b) Beijing (c) London

(d) California (e) Beijing (f) London

Figure 7.9: The results while varying the probability threshold τ

7.5 Summary

In this work, we formalize the probabilistic time-constrained route (PTR) query, which utilizes uncertain road network to describe the uncertainty of the travel time over road network, and retrieves routes that satisfy both keyword and time constraints at POIs, and are in the set of top-h optimal routes with high probabilities. PTR query is particularly useful for users who are touring a city and are unfamiliar with the traffic condition and the service hours of POIs there. In order to efficiently tackle the PTR query problem, we propose three effective pruning strategies to filter out false alarms, and integrate them into an efficient PTR search algorithm. As demonstrated through extensive experiments, the proposed solution achieves satisfying performance over different datasets.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Conducting queries with both spatial and textual requirements has been an important part for many location-aware applications. Existing studies on spatial keyword query cannot well meet such demand since most of them are conducted based on Euclidean distance and inapplicable in the context of road network. Motivated by this, I focus on ***Spatial Keyword Query Processing over Road Networks*** and study three types of queries, i.e., services locating, route planning, and route planning over dynamic road networks.

- **Services locating:** For this type of query, we propose range spatial keyword (RSK) query to search all the objects that are relevant to the query keywords within a given spatial range. RSK can find many applications like searching surrounding services and providing location-based recommendations. To process RSK query, we first propose an expansion-based approach by using the locality property of RSK query. Then, the expansion-based approach is improved based on the observation that network distance is always larger than or equal to the Euclidean distance between two locations. In addition, to increase the scalability of RSK query, we design the Rnet Hierarchy index and devise an efficient processing approach based on this index. As demonstrated by the ex-

periments, Rnet Hierarchy-based approach can efficiently process RSK queries of various query ranges over road networks of millions of vertices.

- **Route planning:** For this type of query, we propose keyword coverage route (KCR) query and bounded-cost informative route (BCIR) query. KCR query searches the optimal route that covers the most number of query keywords within a given distance threshold. BCIR query retrieves the optimal route that is most textually relevant to the query keywords within a given cost budget. KCR is designed to identify routes of many target objects for users to explore while BCIR query is good at finding routes of specific topics and extracting specific travel routes for tourism promotion. To process KCR query, we propose an adaptive route sampling framework and design both static and dynamic route sampling techniques under this framework. Particularly, the dynamic route sampling can improve route sampling by learning knowledge from the routes sampled previously, thus reducing the time of getting the routes of high quality. The proposed route sampling framework can flexibly return query results of different qualities according to the given response time limit, thus providing a tradeoff between the query efficiency and the quality of query results. For BCIR query, considering the high complexity of BCIR query and its non-additive property, we design different solutions for different applications scenarios. On the one hand, we design an exact solution with multiple pruning methods for BCIR queries of small cost budgets. On the other hand, we design a time-bounded solution and an error-bounded solution for BCIR queries of large cost budgets, where time-bounded solution initializes a set of candidate routes and keeps refining them until reaching the response time limit; error-bounded solution is adapted from the exact solution by relaxing the pruning requirement to improve the pruning efficiency and can greatly reduce the query

processing time while guaranteeing the quality of query results. BCIR query can be regarded as a generalization of the KCR query. However, BCIR aims to maximize the textual relevance of a route rather than the number of query keywords.

- **Route planning over dynamic road networks:** For this type of query, we propose probabilistic time-constrained route (PTR) query to search the routes that pass a sequence of POIs and have the optimal travel time in high confidence when considering the dynamic traffic over road networks and the service time constraints at POIs. PTR query is particularly useful for users who are touring a new city and are unfamiliar with the traffic condition and the service hours of POIs there. To solve this query problem, we propose a two-phase solution. First, multiple pruning methods are designed based on the time constraints at POIs and the optimality of routes. With these pruning methods, a processing algorithm is proposed to compute a small set of candidate routes. Second, Monte Carlo sampling is used to refine the computed candidate routes to get the final query results. As demonstrated by the experiments, the proposed solution can efficiently solve the PTR query problem and performs better than the existing uncertain route query techniques.

In sum, we aim to provide users an easy way to have a good knowledge of their required services and the corresponding travel routes to these services. Specifically, the range spatial keyword query provides users the information about the surrounding environments; both keyword coverage route query and bounded-cost informative route query retrieve interesting routes for users to explore; and the probabilistic time-constrained route query is particularly useful for route planning when the dynamics of road network and the time constraints of service providers need to be considered.

8.2 Future Work

As discussed in Section 2, the research on spatial keyword query over road networks is still limited. Except for the existing studies and the work in this topic, there are still many spatial keyword queries over road networks remaining unsolved, e.g., similarity join query to retrieve all pairs of similar spatio-textual objects over road networks, and why not query to uncover the reason why some expected objects are not covered by query results. In addition, there are many other directions relevant to spatio-textual data and road networks. Particularly, I listed a few as below.

- **Spatio-Textual Data Extraction**: Though there is already a large amount of explicit spatio-textual data from different sources, more other spatio-textual data is not so obvious. For example, many comments from social networks do not have clear spatial locations which are hidden in the content. In this case, we need to utilize text analysis techniques to uncover the underlying location attributes. Therefore, studying how to extract spatio-textual data from the Web will greatly increase the scale of spatio-textual data and accordingly extend the applications of spatial keyword query.

- **Dynamics in Spatial Keyword Query**: Apart from moving spatial keyword queries over road networks, almost all other existing queries are conducted based on static data. However, the traffic over road network is dynamically changing over time and the working times of service providers (a kind of spatio-textual objects) along the roads are also time-dependent. Existing spatial keyword queries based on static data cannot capture such dynamics. Therefore, it is of high necessity to revisit all existing spatial keyword queries to make them adapt to the dynamics in both traffic and objects.

- **Advanced Map Construction**: In addition to considering the cost such

as distance and travel time when searching travel routes, users usually wish that the returned routes also have some other properties, e.g., safe, quiet, popular and emotionally pleasant. To retrieve such routes, it is important to construct the corresponding map for each property. Since spatio-textual objects, especially the geo-tagged comments and photos, contain abundant information about these properties, it is promising to construct advanced maps from spatio-textual objects by using machine learning and big data analytic techniques. With these constructed maps, almost all existing route queries can be redefined to provide more personalized route querying service.

Bibliography

[1] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. In *SIGMOD*, pages 34–48, 1987.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[3] Michael G.H. Bell. Hyperstar: A multi-path astar algorithm for risk averse vehicle navigation. *Transportation Research Part B: Methodological*, 43(1):97 – 107, 2009.

[4] Kenneth S. Bøgh, Anders Skovsgaard, and Christian S. Jensen. Groupfinder: A new approach to top-k point-of-interest group retrieval. *PVLDB*, 6(12):1226–1229, 2013.

[5] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[6] Xin Cao, Lisi Chen, Gao Cong, Jihong Guan, Nhan-Tue Phan, and Xiaokui Xiao. KORS: keyword-aware optimal route search system. In *ICDE*, pages 1340–1343, 2013.

[7] Xin Cao, Lisi Chen, Gao Cong, Christian S. Jensen, Qiang Qu, Anders Skovsgaard, Dingming Wu, and Man Lung Yiu. Spatial keyword querying. In *ER*, pages 16–29, 2012.

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

[9] Xin Cao, Gao Cong, Tao Guo, Christian S. Jensen, and Beng Chin Ooi. Efficient processing of spatial group keyword queries. *ACM Trans. Database Syst.*, 40(2):13:1–13:48, 2015.

[10] Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384, 2011.

[11] Xin Cao, Gao Cong, Christian S. Jensen, and Man Lung Yiu. Retrieving regions of interest for user exploration. *PVLDB*, 7(9):733–744, 2014.

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

[13] Anthony Chen and Zhaowang Ji. Path finding under uncertainty. *Journal of Advanced Transportation*, 39(1):19–37, 2005.

[14] Bi Yu Chen, William H. K. Lam, Qingquan Li, Agachai Sumalee, and Ke Yan. Shortest path finding problem in stochastic time-dependent road networks with stochastic first-in-first-out property. *IEEE Trans. Intelligent Transportation Systems*, 14(4):1907–1917, 2013.

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

[16] Lei Chen, Xin Lin, Haibo Hu, Christian S. Jensen, and Jianliang Xu. Answering why-not questions on spatial keyword top-k queries. In *ICDE*, pages 279–290, 2015.

[17] Lei Chen, Jianliang Xu, Xin Lin, Christian S. Jensen, and Haibo Hu. Answering why-not spatial keyword top-k queries via keyword adaption. In *ICDE*, pages 697–708, 2016.

[18] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[19] Lu Chen, Yunjun Gao, Kai Wang, Christian S. Jensen, and Gang Chen. Answering why-not questions on metric probabilistic range queries. In *ICDE*, pages 767–778, 2016.

[20] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, pages 277–288, 2006.

[21] Dong-Wan Choi, Chin-Wan Chung, and Yufei Tao. A scalable algorithm for maximizing range sum in spatial databases. *PVLDB*, 5(11):1088–1099, 2012.

[22] Dong-Wan Choi, Chin-Wan Chung, and Yufei Tao. Maximizing range sum in external memory. *ACM Trans. Database Syst.*, 39(3):21:1–21:44, 2014.

[23] Farhana Murtaza Choudhury, J. Shane Culpepper, Timos K. Sellis, and Xin Cao. Maximizing bichromatic reverse spatial and textual k nearest neighbor queries. *PVLDB*, 9(6):456–467, 2016.

[24] Maria Christoforaki, Jinru He, Constantinos Dimopoulos, Alexander Markowetz, and Torsten Suel. Text vs. space: efficient geo-search query processing. In *CIKM*, pages 423–432, 2011.

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

[26] Camila F. Costa, Mario A. Nascimento, José Antônio Fernandes de Macêdo, and Javam C. Machado. Nearest neighbor queries with service time constraints in time-dependent road networks. In *SIGSPATIAL*, pages 22–29, 2013.

[27] Dingxiong Deng, Cyrus Shahabi, and Ugur Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *SIGSPATIAL*, pages 314–323, 2013.

[28] E.M. Dikjstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[29] Christos Faloutsos and Stavros Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. *ACM Trans. Inf. Syst.*, 2(4):267–288, 1984.

[30] Hailin Fang, Pengpeng Zhao, Victor S. Sheng, Jian Wu, Jiajie Xu, An Liu, and Zhiming Cui. Effective spatial keyword query processing on road networks. In *ADC*, pages 194–206, 2015.

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

[32] Kaiyu Feng, Gao Cong, Sourav S. Bhowmick, Wen-Chih Peng, and Chunyan Miao. Towards best region search for data exploration. In *SIGMOD*, pages 1055–1070, 2016.

[33] Rajiv Gandhi, Samir Khuller, and Aravind Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

[34] Yunjun Gao, Xu Qin, Baihua Zheng, and Gang Chen. Efficient reverse top-k boolean spatial keyword queries on road networks. *IEEE Trans. Knowl. Data Eng.*, 27(5):1205–1218, 2015.

[35] Yunjun Gao, Jingwen Zhao, Baihua Zheng, and Gang Chen. Efficient collective spatial keyword query processing on road networks. *IEEE Trans. Intelligent Transportation Systems*, 17(2):469–480, 2016.

[36] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333, 2008.

[37] 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.

[38] Tao Guo, Xin Cao, and Gao Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418, 2015.

[39] Vivek Gupta and Vikram Goyal. Spatio-textual similarity joins using variable prefix filtering. In *IKDD*, pages 120–121, 2015.

[40] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[41] Randolph W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[42] Ramaswamy Hariharan, Bijit Hore, Chen Li, and Sharad Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *SSDBM*, page 16, 2007.

[43] Ming Hua and Jian Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, pages 347–358, 2010.

[44] He Huang and Song Gao. Optimal paths in dynamic networks with dependent random link travel times. *Transportation Research Part B: Methodological*, 46(5):579–598, 2012.

[45] Weihuang Huang, Guoliang Li, Kian-Lee Tan, and Jianhua Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941, 2012.

[46] Yun-Wu Huang, Ning Jing, and Elke A. Rundensteiner. Effective graph clustering for path queries in digital map databases. In *CIKM*, pages 215–222, 1996.

[47] Alexander Kalinin, Ugur Çetintemel, and Stanley B. Zdonik. Interactive data exploration using semantic windows. In *SIGMOD*, pages 505–516, 2014.

[48] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(2):291–308, 1970.

[49] Ali Khodaei, Cyrus Shahabi, and Chen Li. Hybrid indexing and seamless ranking of spatial and textual features of web documents. In *DEXA*, pages 450–466, 2010.

[50] Ken C. K. Lee, Wang-Chien Lee, and Baihua Zheng. Fast object search on road networks. In *EDBT*, pages 1018–1029, 2009.

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

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

[53] Wengen Li, Jiannong Cao, Jihong Guan, Man Lung Yiu, and Shuigeng Zhou. Retrieving routes of interest over road networks. In *WAIM*, pages 109–123, 2016.

[54] Yujiao Li, Weidong Yang, Wu Dan, and Zhipeng Xie. Keyword-aware dominant route search for various user preferences. In *DASFAA*, pages 207–222, 2015.

[55] Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lun Lee, and Xufa Wang. Ir-tree: An efficient index for geographic document search. *IEEE Trans. Knowl. Data Eng.*, 23(4):585–599, 2011.

[56] Xiang Lian and Lei Chen. Trip planner over probabilistic time-dependent road networks. *IEEE Trans. Knowl. Data Eng.*, 26(8):2058–2071, 2014.

[57] Qing Liu, Yunjun Gao, Gang Chen, Baihua Zheng, and Linlin Zhou. Answering why-not and why questions on reverse top-k queries. *VLDB J.*, 25(6):867–892, 2016.

[58] Cheng Long, Raymond Chi-Wing Wong, Ke Wang, and Ada Wai-Chee Fu. Collective spatial keyword queries: a distance owner-driven approach. In *SIGMOD*, pages 689–700, 2013.

[59] Jiaheng Lu, Ying Lu, and Gao Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360, 2011.

[60] Ying Lu and Cyrus Shahabi. An arc orienteering algorithm to find the most scenic path on a large-scale road network. In *SIGSPATIAL*, pages 46:1–46:10, 2015.

[61] Changyin Luo, Junlin Li, Guohui Li, Wei Wei, Yanhong Li, and Jianjun Li. Efficient reverse spatial and textual k nearest neighbor queries on road networks. *Knowl.-Based Syst.*, 93:121–134, 2016.

[62] Siqiang Luo, Yifeng Luo, Shuigeng Zhou, Gao Cong, and Jihong Guan. Disks: A system for distributed spatial group keyword search on road networks. *PVLDB*, 5(12):1966–1969, 2012.

[63] Siqiang Luo, Yifeng Luo, Shuigeng Zhou, Gao Cong, and Jihong Guan. Distributed spatial keyword querying on road networks. In *EDBT*, pages 235–246, 2014.

[64] Xiaobin Ma, Shashi Shekhar, Hui Xiong, and Pusheng Zhang. Exploiting a page-level upper bound for multi-type nearest neighbor queries. In *ACM-GIS*, pages 179–186, 2006.

[65] Elise Miller-Hooks. Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. *Networks*, 37(1):35–52, 2001.

[66] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[67] Yu Nie and Xing Wu. Shortest path problem considering On-Time arrival probability. *Transportation Research Part B*, 43(6):597–613, 2009.

[68] Yu Marco Nie and Xing Wu. Reliable a priori shortest path problem with limited spatial and temporal dependencies. In *Proceedings of the 18 th International Symposium on Transportation and Traffic Theory*, pages 69–195, 2009.

[69] Daniele Quercia, Rossano Schifanella, and Luca Maria Aiello. The shortest path to happiness: recommending beautiful, quiet, and happy routes in the city. In *HT*, pages 116–125, 2014.

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

[71] João B. Rocha-Junior and Kjetil Nørvåg. Top-k spatial keyword queries on road networks. In *EDBT*, pages 168–179, 2012.

[72] S. Samaranayake, S. Blandin, and A. Bayen. A tractable class of algorithms for reliable routing in stochastic networks. In *ISTTT*, 2011.

[73] Mehdi Sharifzadeh, Mohammad R. Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *VLDB J.*, 17(4):765–787, 2008.

[74] Dimitrios Skoutas, Dimitris Sacharidis, and Kostas Stamatoukos. Identifying and describing streets of interest. In *EDBT*, pages 437–448, 2016.

[75] Anders Skovsgaard and Christian S. Jensen. Finding top-k relevant groups of spatial web objects. *VLDB J.*, 24(4):537–555, 2015.

[76] Wouter Souffriau, Pieter Vansteenwegen, Greet Vanden Berghe, and Dirk Van Oudheusden. The planning of cycle trips in province of eat flanders. *Omega*, 39(2):209–213, 2013.

[77] Sen Su, Sen Zhao, Xiang Cheng, Rong Bi, Xin Cao, and Jie Wang. Group-based collective keyword querying in road networks. *Inf. Process. Lett.*, 118:83–90, 2017.

[78] Lu Sun, Wenjun Gu, and Hani Mahmassani. Estimation of expected travel time using the method of moment. *Canadian Journal of Civil Engineering*, 38(2):154–165, 2011.

[79] Yufei Tao, Xiaocheng Hu, Dong-Wan Choi, and Chin-Wan Chung. Approximate maxrs in spatial databases. *PVLDB*, 6(13):1546–1557, 2013.

[80] Subodh Vaid, Christopher B. Jones, Hideo Joho, and Mark Sanderson. Spatio-textual indexing for geographical search on the web. In *SSTD*, pages 218–235, 2005.

[81] C. Verbeeck, P. Versteenwegen, and E.-H. Aghezzaf. An extension of the arc orienteering problem and its applicaiton to cycle trip planning. *Transportation research*, 68:64–78, 2014.

[82] Yu Ting Wen, Kae-Jer Cho, Wen-Chih Peng, Jinyoung Yeo, and Seung-won Hwang. KSTR: keyword-aware skyline travel route recommendation. In *ICDM*, pages 449–458, 2015.

[83] Dingming Wu, Byron Choi, Jianliang Xu, and Christian S. Jensen. Authentication of moving top-k spatial keyword queries. *IEEE Trans. Knowl. Data Eng.*, 27(4):922–935, 2015.

[84] Dingming Wu, Gao Cong, and Christian S. Jensen. A framework for efficient spatial web object retrieval. *VLDB J.*, 21(6):797–822, 2012.

[85] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S. Jensen. Joint top-k spatial keyword query processing. *IEEE Trans. Knowl. Data Eng.*, 24(10):1889–1903, 2012.

[86] Dingming Wu, Man Lung Yiu, and Christian S. Jensen. Moving spatial keyword queries: Formulation, methods, and analysis. *ACM Trans. Database Syst.*, 38(1):7:1–7:47, 2013.

[87] Dingming Wu, Man Lung Yiu, Christian S. Jensen, and Gao Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

[88] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *PVLDB*, 5(5):406–417, 2012.

[89] Xike Xie, Peiquan Jin, Man Lung Yiu, Jiang Du, Christian S. Jensen, and Mingxuan Yuan. Enabling scalable geographic service sharing with weighted imprecise voronoi cells. In *ICDE*, pages 1522–1523, 2016.

[90] Xike Xie, Peiquan Jin, Man Lung Yiu, Jiang Du, Mingxuan Yuan, and Christian S. Jensen. Enabling scalable geographic service sharing with weighted imprecise voronoi cells. *IEEE Trans. Knowl. Data Eng.*, 28(2):439–453, 2016.

[91] Tao Xing and Xuesong Zhou. Reformulation and solution algorithms for absolute and percentile robust shortest path problems. *IEEE Trans. Intelligent Transportation Systems*, 14(2):943–954, 2013.

[92] Tau Xing and Xuesong Zhou. Finding the most reliable path with and without link travel time correlation: A lagrangian substitution based approach. *Transportation Research Part B: Methodological*, 45(10):1660–1679, 2011.

[93] Han Yan, Xiang Cheng, Sen Su, Qiying Zhang, and Jianliang Xu. Authenticated spatio-textual similarity joins in untrusted cloud environments. In *ICPADS*, pages 685–694, 2016.

[94] Lixing Yang and Xuesong Zhou. Constraint reformulation and a lagrangian relaxation-based solution algorithm for a least expected time path problem. *Transportation Research Part B: Methodological*, 59:22–44, 2014.

[95] Bin Yao, Mingwang Tang, and Feifei Li. Multi-approximate-keyword routing in GIS data. In *SIGSPATIAL*, pages 201–210, 2011.

[96] Man Lung Yiu and Nikos Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, pages 443–454, 2004.

[97] 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.

[98] Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony K. H. Tung, and Masaru Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.

[99] Dongxiang Zhang, Beng Chin Ooi, and Anthony K. H. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 521–532, 2010.

[100] Jia-Dong Zhang, Chi-Yin Chow, and Yanhua Li. igeorec: A personalized and efficient geographical location recommendation framework. *IEEE Trans. Services Computing*, 8(5):701–714, 2015.

[101] Li Zhang, Xiaoping Sun, and Hai Zhuge. Density based collective spatial keyword query. In *SKG*, pages 213–216, 2012.

[102] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. G-tree: an efficient index for KNN search on road networks. In *CIKM*, pages 39–48, 2013.

[103] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162, 2005.

[104] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.