Answering why-not questions on spatial keyword top-k queries /Chen Lei.

Lei Chen

Follow this and additional works at: https://repository.hkbu.edu.hk/etd_oa

Recommended Citation
https://repository.hkbu.edu.hk/etd_oa/365

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at HKBU Institutional Repository. It has been accepted for inclusion in Open Access Theses and Dissertations by an authorized administrator of HKBU Institutional Repository. For more information, please contact repository@hkbu.edu.hk.
DATE: December 13, 2016

STUDENT'S NAME: CHEN Lei

THESIS TITLE: Answering Why-Not Questions on Spatial Keyword Top-k Queries

This is to certify that the above student's thesis has been examined by the following panel members and has received full approval for acceptance in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Chairman: Dr. Tong Tiejun
Associate Professor, Department of Mathematics, HKBU
(Designated by Dean of Faculty of Science)

Internal Members: Prof. Ng Joseph K Y
Professor, Department of Computer Science, HKBU
(Designated by Head of Department of Computer Science)

Dr. Chen Lisi
Assistant Professor, Department of Computer Science, HKBU

External Members: Dr Cong Gao
Associate Professor
School of Computer Science and Engineering
Nanyang Technological University
Singapore

Dr. Yiu Man Lung
Associate Professor
Department of Computing
The Hong Kong Polytechnic University

In-attendance: Prof. Xu Jianliang
Professor, Department of Computer Science, HKBU

Issued by Graduate School, HKBU
Answering Why-Not Questions on Spatial Keyword Top-\(k\) Queries

CHEN Lei

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Principal Supervisor: Prof. XU Jianliang

Hong Kong Baptist University

December 2016
DECLARATION

I hereby declare that this thesis represents my own work which has been done after registration for the degree of PhD at Hong Kong Baptist University, and has not been previously included in a thesis or dissertation submitted to this or any other institution for a degree, diploma or other qualifications.

I have read the University’s current research ethics guidelines, and accept responsibility for the conduct of the procedure in accordance with the University’s Committee on the Use of Human & Animal Subjects in Teaching and Research (HASC). I have attempted to identify all the risks related to this research that may arise in conducting this research, obtained the relevant ethical and/or safety approval (where applicable), and acknowledged my obligations and the rights of the participants.

Signature: CHEN

Date: December 2016
Abstract

With the continued proliferation of location-based services, a growing number of web-accessible data objects are geo-tagged and have text descriptions. Spatial keyword top-$k$ queries retrieve $k$ such objects with the best score according to a ranking function that takes into account a query location and query keywords. However, it is in some cases difficult for users to specify appropriate query parameters. After a user issues an initial query and gets back the result, the user may find that some expected objects are missing and may wonder why. Answering the resulting why-not questions can aid users in retrieving better results and thus improve the overall utility of the query functionality. While spatial keyword querying has been studied intensively, no proposals exist for how to offer users explanations of why such expected objects are missing from results. In this dissertation, we take the first step to study the why-not questions on spatial keyword top-$k$ queries. We provide techniques that allow different revisions of spatial keyword queries such that their results include one or more desired, but missing objects. Detailed problem analysis and extensive experimental studies consistently demonstrate the effectiveness and robustness of our proposed techniques in a broad range of settings.

Keywords: Spatial Keyword Top-$k$ Query, Why-Not Question, Query Refinement
List of Publications


Acknowledgements

This thesis becomes a reality with the kind support and help of many individuals. I would like to extend my sincere thanks to all of them.

Foremost, I would like to express my sincere gratitude to my supervisor, Prof. Jianliang Xu, for his continuous support of my Ph.D. study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and completing this thesis. My sincere thanks also go to Prof. Christian S. Jensen, Dr. Byron Choi and Dr. Haibo Hu for their encouragement, insightful comments and constructive suggestions during my Ph.D. study.

I also would like to thank my fellow labmates in HKBU Database Group for their direct and indirect help, for the stimulating discussions, and for all the fun we have had in the last four years. In particular, I should mention Dr. Yafei Li, Dr. Qian Chen, Dr. Zhe Fan, Mr. Peipei Yi, Dr. Jingjing Chen, Mr. Cheng Xu, Dr. Yun Peng, Dr. Xin Lin, Dr. Rui Chen, Dr. Dingming Wu, Dr. Qilong Han, Mr. Xiaojing Xie, Ms. Shenshen Peng, Mr. Zhuo Chen, Ms. Li Chen, Mr. Jiaxin Jiang, Ms. Xiaoyi Fu, Mr. Jinbing Huang among many others.

Last but not the least, I take this special occasion to thank my father Huaigen Chen and my mother Aiqun Shu for unconditionally raising me, supporting me and providing me everything as best as they can throughout my life. I also wish to thank my heart dear He Liu for her love, kindness and understanding. Without them, I would never go so far.
Table of Contents

Declaration i
Abstract ii
List of Publications iii
Acknowledgements iv
Table of Contents v
List of Tables ix
List of Figures x

Chapter 1 Introduction 1
  1.1 Preference-Adjusted Why-Not Spatial Keyword Top-k Query . . . . . 2
  1.2 Keyword-Adapted Why-Not Spatial Keyword Top-k Query . . . . . 3
  1.3 Direction-Aware Why-Not Spatial Keyword Top-k Query . . . . . 5
  1.4 Summary ................................................................. 6

Chapter 2 Literature Review 7
  2.1 Spatial Keyword Query Processing ........................................ 7
  2.2 Why-Not Query Processing ............................................. 9

Chapter 3 Preference-Adjusted Why-Not Spatial Keyword Top-k Query 11
  3.1 Preliminaries and Problem Definitions ................................. 11
3.1.1 Spatial Keyword Top-\(k\) Query .................................. 11
3.1.2 Preference-Adjusted Why-Not Spatial Keyword Query ............ 13
3.1.3 Baseline Algorithm .................................................. 14
3.2 Properties and Basic Algorithm ........................................ 16
3.2.1 Projection of Objects .................................................. 16
3.2.2 Query Modification and Ranking Updates ........................... 18
3.2.3 Basic Algorithm for Why-Not Spatial Keyword Query .......... 20
3.3 Bound and Prune Algorithm ............................................ 21
3.3.1 A Hybrid Index: BIR-Tree .......................................... 22
3.3.2 Optimized Query Processing by Estimated Bounds ............... 24
3.3.3 Derivation of \(\hat{D}P(N,e)\) and \(\check{D}P(N,e)\) ...................... 25
3.4 Handling Multiple Missing Objects .................................... 33
3.4.1 Identifying Working Parts .......................................... 34
3.4.2 Why-Not Spatial Keyword Query Processing for Multiple Miss-
ing Objects ................................................................. 36
3.5 Empirical Study ......................................................... 37
3.5.1 Experimental Setup .................................................. 37
3.5.2 Empirical Results ..................................................... 39
3.6 Summary ........................................................................ 43

Chapter 4  Keyword-Adapted Why-Not Spatial Keyword Top-\(k\) Query 44

4.1 Preliminaries and Problem Definition ................................... 44
4.1.1 Preliminaries .............................................................. 44
4.1.2 Keyword-Adapted Why-Not Spatial Keyword Query ............ 45
4.2 Basic Why-not Answering Algorithm .................................... 48
4.2.1 Problem Analysis ....................................................... 48
4.2.2 Basic Algorithm ......................................................... 48
4.2.3 Optimizations ............................................................ 50
4.3 Bound-and-Prune Algorithm ............................................. 53
4.3.1 Preliminaries: KcR-Tree .............................................. 55
4.3.2 Properties of KcR-tree ........................................... 56
4.3.3 Optimized Bound-and-Prune Algorithm ..................... 60
4.3.4 Strategic Use of Algorithm 5 ................................. 61
4.4 Multiple Missing Objects and Approximate Algorithm ....... 64
  4.4.1 Multiple Missing Objects ................................. 64
  4.4.2 Approximate Algorithm ................................. 65
4.5 Empirical Study .................................................. 65
  4.5.1 Experimental Setup .................................. 66
  4.5.2 Experimental Result .................................. 67
4.6 Summary ......................................................... 74

Chapter 5  Direction-Aware Why-Not Spatial Keyword Top-k Query 76

  5.1 Preliminaries and Problem Formulation ....................... 76
    5.1.1 Preliminaries .................................... 77
    5.1.2 Direction-Aware Why-Not Spatial Keyword Query .... 79
    5.1.3 Baseline Algorithm .................................. 80
  5.2 Answering Why-Not Questions: Special Case ............... 81
    5.2.1 Case Analysis .................................... 82
    5.2.2 Ranking Updates and Direction Modification .......... 82
    5.2.3 Answering Why-Not .................................. 85
  5.3 Answering Why-Not Questions: General Case ............... 88
    5.3.1 Case Analysis .................................... 88
    5.3.2 Answering Why-Not .................................. 92
  5.4 Handling Multiple Missing Objects .......................... 95
  5.5 Empirical Study .............................................. 97
    5.5.1 Experimental Setup .................................. 98
    5.5.2 Experimental Results .................................. 99
  5.6 Summary ...................................................... 105
List of Tables

3.1 An Example of Refined Queries (\(\lambda = 0.5\)) ........................................ 14
3.2 Summary of notations ................................................................. 16
3.3 Relations between Node \(N\) and Object \(o\) ................................. 28
3.4 Dataset Characteristics ......................................................... 38
3.5 Parameter Settings ................................................................. 39

4.1 An Example of Refined Queries ................................................ 47
4.2 Dataset Information ................................................................. 66
4.3 Parameter Setting ................................................................. 67

5.1 Dataset Information ................................................................. 99
5.2 Parameter Setting ................................................................. 99
List of Figures

2.1 An Example IR-tree ............................................. 8

3.1 An Example of Why-not Spatial Keyword Query .................. 14
3.2 An Example of Fixed-Score Segment .............................. 15
3.3 An Example of Weighting Vector Rotation and Ranking Updates .. 18
3.4 Structure of a BIR-Tree ........................................ 22
3.5 An Example of the Boundary Region of a Node $N$ .............. 27
3.6 An Example of $\#$ Degraded Point Estimation .................... 29
3.7 An Example of Multiple Missing Objects .......................... 33
3.8 Varying Dataset Size ......................................... 39
3.9 Varying $k_0$ ................................................ 40
3.10 Varying Number of Query Keywords ............................. 41
3.11 Varying Initial Weighting Vector $\vec{w}_0$ ......................... 41
3.12 Varying $\lambda$ ................................................ 42
3.13 Varying Number of Missing Objects ............................. 43

4.1 An Example of Why-not Spatial Keyword Top-$k$ Query ........ 47
4.2 An Example SetR-tree ........................................ 49
4.3 An Example KcR-tree .......................................... 56
4.4 Varying $k_0$ ................................................ 68
4.5 Varying the number of initial query keywords .................... 69
4.6 Varying $\alpha$ ............................................... 69
4.7 Varying $\lambda$ ............................................... 70
4.8 Varying the missing object’s initial ranking ...................... 71
4.9 Varying the number of missing objects .................................. 72
4.10 Varying the number of threads .............................................. 72
4.11 Effects of different optimizations ......................................... 72
4.12 Varying Sample Size T ......................................................... 73
4.13 Varying dataset size ......................................................... 74

5.1 Top k and Direction-Aware Top-k Spatial Keyword Query ........ 79
5.2 A candidate end (start) direction for a given $\alpha'$ ($\beta'$) ........ 84
5.3 Candidate directions for a given refined $k'$ ............................. 88
5.4 Cases of the reason about $m$’s missing ................................. 89
5.5 Ranges of $\alpha$ and $\beta$ that rank $m$ at 2 ................................. 94
5.6 An example of multiple missing objects ................................. 97
5.7 Running time vs. Baseline & Algorithm 7 .............................. 100
5.8 Running time vs. $k_0$ ......................................................... 101
5.9 Running time vs. $\#$ keywords ........................................... 102
5.10 Running time vs. $w_s$ in $\bar{w}$ .......................................... 102
5.11 Running time vs. $d_0$ ......................................................... 103
5.12 Running time vs. Rank of missing object .............................. 104
5.13 Running time vs. $\#$ missing objects ................................. 104
5.14 Running time vs. dataset size ............................................ 105
Chapter 1

Introduction

With the proliferation of geo-enabled mobile devices, notably smartphones, location-based services that target web objects with geo-location and textual descriptions, e.g., businesses and public facilities, are gaining in prominence. In particular, so-called spatial keyword queries [3] enable a range of services that retrieve such objects. More specifically, a spatial keyword top-$k$ query takes a user location and a set of keywords as arguments and retrieves the $k$ objects that are ranked the highest according to a scoring function that considers both spatial distance and textual similarity [14].

However, it can be difficult for users to specify the appropriate parameters that best capture the intent of their queries. Thus, after a user issues a spatial keyword top-$k$ query and receives the result, the user may find that the result is not as expected. Specifically, objects that the user expected to be in the result are missing. This suggests to the user that other useful objects, that are as yet unknown to the user, may be missing from the result as well, and the user has reason to question the overall utility of the query and its result.

Debugging and fixing a query consumes time and may require insight that a user does not have. The utility of spatial keyword querying can be improved by offering functionality that explains to users why one or more expected objects are missing and how to minimally modify the initial query so that the missing objects, and then potentially also other useful objects, become part of the result.
Several approaches may be taken to support such why-not questions, including manipulation identification [6], database modification [20, 21], and query refinement [19, 36]. Chapman and Jagadish [6] study why-not functionality for Select-Project-Join (SPJ) queries that is able to determine the query operator that prevents a missing object from being included in a result. Other works [20, 21] study how to update a database so that SPJ queries and SPJUA (SPJ + Union + Aggregation) queries revive missing objects. Early work [36] using the query refinement approach studies how to revise an original query so that a missing object enters the result. He and Lo [19] adapted this approach to top-$k$ preference queries and study how to minimize the overall change of weights in the ranking function and the parameter $k$ while achieving the inclusion. However, no existing work study the problem of answering why-not questions on spatial keyword top-$k$ queries. In this dissertation, we adopt the query refinement model to answer why-not questions on spatial keyword top-$k$ queries considering the refinement of different parameters to support diverse application scenarios.

1.1 Preference-Adjusted Why-Not Spatial Keyword Top-$k$ Query

The spatial keyword top-$k$ query ranks objects considering both spatial distance and textual similarity. A common approach is to combine these two factors using a linear model, where a weighting vector is used to balance their importance in the overall ranking score. However, it is difficult for end users to measure their own preference weighting vectors and it can be inapplicable to let end users specify the weighting vectors, as the ranking function is often supposed to be a black-box to them. Another way is to leave the weighting vector as a system default. This is also not applicable as users do have different preferences over these two factors.

Example 1. Bob visits New York for the first time, and he wants to find a nearby cafe for a cup of coffee. He issues a top-10 spatial query with keyword “coffee.”
However, surprisingly, the Starbucks cafe down the street not far away, is not in the result. Bob wonders why the Starbucks cafe is not in the result? Are there better options? Is something wrong with the query so that other good options are also missing? How can the ranking function be adjusted so that the Starbucks cafe, and perhaps other relevant cafes, appears on the result?

As illustrated by the example, the expected Starbucks cafe may be missing from the result due to an improper setting on the preference weighting vector. The first part of the dissertation is devoted to helping users adjust the preferences between spatial proximity and textual relevance to achieve the inclusion of their desired but missing object in query results. To efficiently answer a preference-adjusted why-not question for a query, we project the objects and weighting vectors onto a two-dimensional plane and obtain a geometrical model for similarity ranking. Under this model, a modification of weights is equivalent to a rotation of a weighting vector. As such, an update to the ranking for a missing object can be converted to a two-dimensional geometrical problem. We prove that the best weighting vector derives from a finite set of candidate vectors. To further optimize the performance, we propose an index-based ranking estimation algorithm, which prunes candidate weighting vectors early during index traversal. In addition, we extend the proposed algorithms to support why-not spatial keyword queries for multiple missing objects. Chapter 3 is dedicated to answering such preference-adjusted why-not spatial keyword top-\(k\) queries.

1.2 Keyword-Adapted Why-Not Spatial Keyword Top-\(k\) Query

Considering that it can also be difficult for users to identify the keywords that best capture the intent of their queries, the missing of expected object may due to inappropriate query keywords. The second part of the dissertation is devoted to providing users with more accurate query keywords to get the inclusion of the
desired but missing objects in the query result. The motivation and significance of
this functionality is illustrated by the following two examples.

Example 2. In preparation for attending an overseas conference, a user issues a
query to find the top-3 hotels that are close to the conference venue and are described
as “clean” and “comfortable.” The user is surprised that the result contains only
local hotels that are unknown to the user and that a well-known nearby international
hotel is not in the result. The user wonders why this exclusion happens. Are the
returned hotels really the best, or are there better options? Are the query keywords
not properly set? How can the keywords be adapted so that the expected hotels
appear in the result?

Another scenario is for merchant users who want to refine a set of keywords to
best advertise their products.

Example 3. Consider a user who opens a Sichuan restaurant near the Oriental
Pearl Tower in Shanghai, China. The user lists the restaurant in online catalogs such
as Google My Business. To attract more customers, the user wants to advertise the
keywords in which the restaurant ranks high when the customers search the catalog
near the Oriental Pearl Tower. However, the simple keywords “Sichuan cuisine” do
not return the restaurant as a top-10 result. The user wants to know why and how
the keywords should be modified so that the restaurant can enter the top-10.

In Chapter 4, we formulate the keyword-adapted why-not spatial keyword top-k
query and reduce it to a query refinement problem. We propose a basic algorithm as
well as a set of optimizations to efficiently search the candidate query keyword sets
to find the optimal solution. In addition, we propose a more advanced algorithm
based on an index structure, which determines the best keyword set among a set of
candidates using a bound-and-prune strategy. Both algorithms are also extended to
solve the why-not problems with multiple missing objects.
1.3 Direction-Aware Why-Not Spatial Keyword Top-$k$ Query

While the spatial keyword top-$k$ query has been studied extensively, it is noted that direction-aware search is demanded in many LBS scenarios [25]. A direction-aware spatial keyword query aims to retrieve the top-$k$ objects that best match query parameters in terms of spatial distance and textual similarity in a given query direction. However, there may be cases where users are not fully aware of the appropriate direction to feed to a spatial keyword top-$k$ query; it may be difficult for a user to specify the direction that best captures the intent of her query. The expected object is missing from a query result may due to the improper setting on the direction, as illustrated in the following example.

**Example 4.** After a busy day of sightseeing, Clair is tired and hungry. Walking back to the hotel, she issues a query to find the top-3 nearby “Sushi” restaurants. Surprisingly, she finds that the result contains only restaurants that are out of his way and that a restaurant on her way to the hotel that she visited yesterday is not in the result. Clair questions the overall result. Are the returned restaurants really the best, or do better options exist? Should the restaurants be searched in the general direction of her way to the hotel? How can she add a search direction so that the missing restaurant and possibly other good options appear in the result?

The third part of the dissertation is devoted to providing users with more precise query directions. We aim to minimally modify users initial queries to reintroduce the expected but missing objects into the query result. To this end, we consider the problem in both the special case, where the initial query is a traditional spatial keyword top-$k$ query without a query direction, and the general case, where a query direction is specified initially. To achieve an efficient solution to this problem, we prove that the best refined query direction lies in a finite solution space for the special case, and we reduce the search for the best refined queries into solving linear programming problems for the general case. Furthermore, we extend proposed
algorithms to support why-not questions with multiple missing objects. Chapter 5 presents the detailed discussion on answering such direction-aware why-not spatial keyword top-k queries.

1.4 Summary

The rest of the dissertation is organized as follows. Chapter 2 presents the literature review. Chapter 3, Chapter 4, and Chapter 5 present (i) the problem formulation, (ii) the problem analysis, (iii) the proposed algorithms and (iv) the experimental results on answering preference-adjusted, keyword-adapted, direction-aware why-not spatial keyword top-k queries, respectively. Chapter 6 concludes the dissertation.
Chapter 2

Literature Review

To the best of our knowledge, no studies exist that consider why-not questions on spatial keyword top-$k$ queries. In this section, we survey studies on spatial keyword queries and why-not queries separately, and we distinguish them from the setting of why-not spatial keyword queries.

2.1 Spatial Keyword Query Processing

A spatial keyword query retrieves the most relevant spatial web objects with respect to both spatial distance and textual similarity. Numerous studies have considered this query, and a number of indexing and efficient query processing techniques have been proposed. Early work presents a hybrid index structure that integrates R*-tree and inverted file for the estimation of both spatial and textual similarities [42]. Martins et al. [31] suggest computing text relevancy and location proximity independently and then combine the two aspects. Rocha-Junior and Nørvåg [33] study the spatial keyword query in road networks. Another hybrid index, the IR$^2$-tree [15] combines an R-tree [18] with superimposed text signatures. However, this index is applicable only when the keywords serve as a Boolean filter. The IR-tree [14, 26, 37] is the most widely used index for processing spatial keyword top-$k$ queries. Fig. 2.1 illustrates an example of the IR-tree, where (a) shows the distributions of the indexed objects in space and (b) shows the structure of the IR-tree. Essentially, the
IR-tree is an R-tree, where each node is enriched with an inverted file for the objects indexed in the subtree of this node. The inverted file contains a vocabulary of all distinct terms in the subtree, and for each term $t$, it stores the maximum weight of the term among all the objects in the subtree. As such, the IR-tree is able to estimate the bounds of spatial distance and textual similarity at the same time.

After indexing the spatial textual objects using an IR-tree, the processing for a spatial keyword top-$k$ query is then reduced to a traversal of the IR-tree. With the best-first traversal strategy, they maintain a priority queue $Q$, which is initialized with the IR-tree root node, to track the nodes and objects to be visited. Each time, they dequeue an element from $Q$. The element is returned as a result if it is an object; otherwise, they estimate the bounds for its children and insert them into $Q$. The process continues until $k$ objects are returned. A comprehensive experimental evaluation of different spatial keyword indexing and query processing techniques is available [7].

Different variants of the spatial keyword query have also been considered. Chen et al. [12] study a query that retrieves web pages which contain query keywords and whose page footprints intersect with a query footprint. A recently proposed $m$CK query retrieves the best $m$ objects within a minimum diameter that match given keywords. The bR*-tree and the virtual bR*-tree [40, 41], which augment each node with a bitmap and MBRs for keywords, are proposed for computing this query. Cao et al. [4] introduce a query that retrieves the top-$k$ spatial web objects ranked according to both prestige-based relevance and location proximity. Another
study [5] proposes a query that retrieves a group of nearby spatial web objects whose keywords cover the query’s keywords and that have the lowest inter-object distances. Further, spatial keyword similarity search in regions of interest has been studied [16]. Li et al. [27] investigates a spatial approximate string query that is a range query augmented with a string similarity predicate. Bouros et al. [2] aim to identify pairs of objects from a spatio-textual database that are both spatially close and textually similar. Another study [38] integrates the social influence into traditional spatial keyword search to improve the answer quality. More recently, Lee et al. [24] study the processing and optimizations for main memory spatial keyword queries. Choudhury et al. [13] aim to find an optimal location and a set of keywords that maximize the size of bichromatic reverse spatial textual $k$ nearest neighbors. Shi et al. [34] study location-based keyword search on RDF data. Considering that the query direction is demanded in many real-life scenarios, Li et al. study the direction-aware spatial keyword query that finds $k$ nearest neighbors that cover all query keywords in the search direction [25].

However, none of the work mentioned above address the why-not spatial keyword query problem.

### 2.2 Why-Not Query Processing

After decades of effort working, the capability of database systems in terms of both performance and functionality has achieved great improvements [23]. Recently, people in the database community notice that the usability of a database is as important as its capability. To improve the usability of database systems, the concept of why-not problem was introduced by Chapman and Jagadish [6], which aims to make users be able to question why particular data items not show up in the result of an issued query and thus make the database system more interactive and user-friendly. Afterwards, database researchers make great efforts to study why-not questions on different queries. Existing approaches can be classified into three categories: (1) manipulation identification [6], which identifies query operators that prevent miss-
ing objects from being included in a result; (2) database modification [20, 21], which updates the original database so that the query can revive missing objects; and (3) query refinement [19, 36], which revises the original query so that missing objects can enter the result.

As the first, Chapman and Jagadish use manipulation identification to identify operations that filter out expected missing objects on Select-Project-Join (SPJ) queries. The proposed techniques in this work are designed for relational database and are able to help users better understand the data behind and further debug their queries. Other studies [20, 21] adopt a database modification to update an original database so that missing objects become part of query results. Tran and Chan [36] retrieve the missing objects through query refinement on SPJ+Aggregation queries. He and Lo [19] employ the query refinement model to answer why-not questions on top-\(k\) preference queries. More recent, studies consider how to answer why-not questions using query refinement in the contexts of social image search [1], reverse skyline queries [22], reverse top-\(k\) queries [17] and metric probabilistic range queries [10], respectively. As many different modifications on the initial query could achieve the inclusion of the missing object, they propose to find the refined query with the minimal changes to the origin query. For example, in [19], the authors aim to modify the original preference weighting vector \(\tilde{w}\) and the result set cardinality \(k\) with the minimum penalty to revive the missing object in the top-\(k\) preference query. They prove that the candidates for the best refined preference weighting vector lie in a much smaller space than the whole search space. Nevertheless, the candidate space still contains infinite number of possible refined weighting vectors. They propose to sample part of the candidate weighting vectors; then for each of them, process the top-\(k\) preference query until the expected object appears in the result to determine the corresponding refined \(k\); and finally identify the pair of refined \(k\) and \(\tilde{w}\) with the smallest penalty to get an approximate solution.

However, the why-not question has not been studied for spatial keyword queries. The existing techniques are not applicable here.
Chapter 3

Preference-Adjusted Why-Not Spatial Keyword Top-$k$ Query

In this chapter, we investigate the problem of preference-adjusted why-not spatial keyword top-$k$ query. Specifically, Section 3.1 describes some preliminaries and defines the why-not spatial keyword query. Section 3.2 covers how to convert the ranking update to a geometric problem and proposes a basic query processing algorithm. Section 3.3 proposes an index-based optimization and Section 3.4 extends the problem to handle multiple missing objects. The experimental results are presented in Section 3.5. We conclude this chapter in Section 3.6.

3.1 Preliminaries and Problem Definitions

We first define the problem of answering why-not questions on spatial keyword queries. Then we present an analysis of the problem, followed by a baseline algorithm.

3.1.1 Spatial Keyword Top-$k$ Query

Let $\mathcal{D}$ denote a database of spatial web objects. Each object $o$ in $\mathcal{D}$ is denoted by a pair $(o.loc, o.doc)$, where $o.loc$ is a multi-dimensional point location and $o.doc$
is a text document. A spatial keyword top-$k$ query retrieves $k$ top ranked objects according to a scoring function that considers both spatial distance and textual similarity to a query $q$. We adopt the scoring function in [14] as follows:

$$ST(o, q, \vec{w}) = ws \cdot (1 - SDist(o, q)) + wt \cdot TSim(o, q),$$  \hspace{1cm} (3.1.1)$$

where $SDist(o, q)$ denotes spatial distance normalized to a value between 0 and 1 by dividing the Euclidean distance by the maximum possible distance between two objects in the data space, $TSim(o, q)$ denotes textual similarity, and $\vec{w} = (ws, wt)$, where $0 < ws, wt < 1$ and $ws + wt = 1$, is a weighting vector on the relative preference between spatial distance and textual similarity.

As such, a query $q$ is a 4-tuple $(loc, doc, k, \vec{w})$, where $q.loc$ is a query point location; $q.doc$ is a set of keywords; $q.k$ is the number of objects to retrieve; and $\vec{w}$ is the weighting vector. Without loss of generality, we assume no two objects or queries are located at the same point, or apart with the maximum possible distance, and we further assume all scores are unique. Next, the textual similarity $TSim(o, q)$ can be computed using an information retrieval model [30], such as the language model, cosine similarity, or BM25, and is also normalized. We adopt the language model, so the range of $TSim(o, q)$ is $(0,1)$. In the ranking function, the higher the score computed by Eqn. 3.1.1, the higher the rank of the corresponding object. Objects that rank higher than $o$ are called $o$’s dominators under the given weighting vector $\vec{w}$, and we define the rank of an object as follows:

$$R(o, q, \vec{w}) = |\{o' \in D \mid ST(o', q, \vec{w}) > ST(o, q, \vec{w})\}| + 1.$$  \hspace{1cm} (3.1.2)$$

With the definition of ranking, the spatial keyword top-$k$ query is defined as follows:

**Definition 3.1.1. Spatial Keyword Top-$k$ Query.** A spatial keyword top-$k$ query $q$ returns $k$ objects in $D$ that maximize the scoring function in Eqn. 3.1.1, or in terms of ranking, it returns the object set $\{o \mid R(o, q, \vec{w}) \leq k\}$. 

3.1.2 Preference-Adjusted Why-Not Spatial Keyword Query

When a user issues a spatial keyword top-$k$ query $q = (loc, doc, k_0, \bar{w}_0)$, the user may observe that one or more objects that were expected to be in the result are missing. The user may then pose a why-not query with a set of missing objects $M = \{o_1, o_2, ..., o_j\}$, asking the system to identify and process a refined spatial keyword query $q' = (loc, doc, k', \bar{w}')$ that has a result that contains the missing objects. In devising the refined query, we consider the modification of the parameters $k$ and $\bar{w}$ in the original query. A naive approach is to increase $k$ until all missing objects appear in the result. This is not a good approach; rather, we need to evaluate the quality of a refined query against the original query. In doing so, we adopt the penalty model proposed in [19], which uses $\Delta k$ and $\Delta w$ to measure the degree of modification with respect to the original query, where $\Delta k = \max(0, k' - k_0)$ and $\Delta w = \|\bar{w}' - \bar{w}_0\|_2$. Based on this, the penalty (i.e., cost) of a refined query from $q$ is defined as follows:

$$Penalty(k', \bar{w}') = \lambda \cdot \frac{\Delta k}{R(o, q, \bar{w}_0) - k_0} + (1 - \lambda) \cdot \frac{\Delta w}{\sqrt{1 + ws_0^2 + wt_0^2}},$$

(3.1.3)

where $\lambda \in (0, 1)$ is a user preference of the modification on $k$ and $\bar{w}$ from the initial query. In the best refined query with weighting vector $\bar{w}'$, if $R(o, q, \bar{w}') > k_0$, $k'$ should be equal to $R(o, q, \bar{w}')$ to achieve the lowest penalty; otherwise, $k'$ does not need to be modified. So $\Delta k = \max(0, R(o, q, \bar{w}') - k_0)$. We normalize $\Delta k$ and $\Delta w$ by dividing them by $R(o, q, \bar{w}_0) - k_0$ and $\sqrt{1 + ws_0^2 + wt_0^2}$, respectively. As proved in [19], $\Delta k$ in the best refined query is no larger than $R(o, q, \bar{w}_0) - k_0$, and $\Delta w$ is no larger than $\sqrt{1 + ws_0^2 + wt_0^2}$.

Based on the above, we formally define the why-not spatial keyword query as follows:

**Definition 3.1.2. Preference-Adjusted Why-Not Spatial Keyword Query.**

*Given an object set $\mathcal{D}$, a missing object set $M \subset \mathcal{D}$, an original spatial keyword query $q: (loc, doc, k_0, \bar{w}_0)$, the preference-adjusted why-not spatial keyword query returns*
the refined query with the lowest penalty according to Eqn. 3.1.3, which includes all objects in $M$ in the query result.

Fig. 3.1 exemplifies the why-not spatial keyword query. Here (a) shows the spatial locations of the query and objects, and (b) lists the $1 - SDist(o, q)$ and $TSim(o, q)$ values of the objects in $D$. In the original query, $k_0 = 1$, $w_0 = (0.5, 0.5)$. Object $o$ has $R(o, q, \vec{w}_0) = 3$, so it is missing from the top-1 result. Table 3.1 shows some refined queries together with their penalty values, where $\lambda = 0.5$. According to this setting, $R(o, q, \vec{w}_0) - k_0 = 2$ and $\sqrt{1 + ws_0^2 + wt_0^2} = 1.22$. It is clear that the refined query $q_1$, the only modification of which is to set $k = 3$, is the refined query with the lowest penalty.

### 3.1.3 Baseline Algorithm

The idea of the baseline algorithm is as follows. After the user specifies the initial spatial keyword top-$k$ query, the spatial distance and textual similarity between the query point and each object in $D$ become two constant values. That is, each object can be represented by a vector $\vec{v} = (1 - SDist(o, q), TSim(o, q))$, and the
scoring function becomes $ST(o, q, \bar{w}) = \bar{v} \cdot \bar{w}$. As such, the problem is reduced to “answering why-not questions on 2-dimensional top-$k$ queries,” which can be solved by an existing algorithm [19] as follows (for now we only consider a single missing object). We first calculate the vector $\bar{v}$ for each object in $D$ and find the set $I$ of objects in $D$ that are incomparable with the missing object $o$ using an algorithm introduced in [35], where “incomparable” means that one of two objects is closer to the query point than the other, while the other has a higher textual similarity. Then we maintain a weighting vector set $S$ containing the initial weighting vector $\bar{w}_0$ and the weighting vector $\bar{w}_i$ for each incomparable object $i$ that satisfies $\bar{v}_i \cdot \bar{w}_i = \bar{v}_o \cdot \bar{w}_i$.

Finally, for each weighting vector $\bar{w}_i$ in $S$, we issue a top-$k$ query to obtain the ranking of the missing object under $\bar{w}_i$. The one with the lowest penalty is returned as the result. This algorithm can be adapted to handle multiple missing objects.

However, accessing the whole database and then computing all these distances and similarities at runtime is very time-consuming. Furthermore, for each incomparable object of the missing object $o$, a top-$k$ query needs to be issued. Therefore, the time complexity is proportional to the number of incomparable objects, which can be high.
Table 3.2: Summary of notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R(o,q,\vec{w})$</td>
<td>The ranking of object $o$ under query $q$ and weighting vector $\vec{w}$.</td>
</tr>
<tr>
<td>$[\vec{w}_1,\vec{w}_2]$</td>
<td>The angle interval from $\vec{w}_1$ to $\vec{w}_2$.</td>
</tr>
<tr>
<td>$l_{\vec{w}}$</td>
<td>The line of weighting vector $\vec{w}$.</td>
</tr>
<tr>
<td>$p(l_1,l_2)$</td>
<td>The intersection point of lines (or segments) $l_1$ and $l_2$.</td>
</tr>
<tr>
<td>$S_o$</td>
<td>The fixed-score segment of object $o$.</td>
</tr>
<tr>
<td>$S_o^s$ or $S_o^t$</td>
<td>The intersection point between segment $S$ and the $ws$-axis or $wt$-axis.</td>
</tr>
</tbody>
</table>

### 3.2 Properties and Basic Algorithm

We now consider the case of a single missing object and introduce a basic processing algorithm. In Section 3.4, we consider the case of multiple missing objects.

We first use the transformation introduced by Chester et al. [11] to project the objects in $D$ and the weighting vectors onto a two-dimensional plane, which visualizes the objects’ rankings when varying the weighting vectors. Then we model the modification of a weighting vector by the rotation of the vector. Under this modification model, the ranking updates of a missing object have a two-dimensional, geographical interpretation. By knowing the initial ranking and the ranking update rule, the basic algorithm finds the best refined weighting vector without exhaustively enumerating all weighting vectors. Table 3.2 summarizes notations used in the following.

#### 3.2.1 Projection of Objects

From the earlier analysis, after the user specifies an initial spatial keyword top-$k$ query, the spatial and textual proximities between an object $o$ and the query point
Proposition. We assume $\mathbf{vt}$, let $ST$ line of score segments. Theorem 3.2.2. According to the definition of the weighting vector the initial query, all objects can be represented by fixed-score segments. Further, the weighting vector $\mathbf{v}$ in the query can also be projected into the “ws–vt” plane. According to the definition of $\mathbf{v}$, it must be a vector from the origin $O$ to some point on $nw$, where $nw$ is the segment from $(1,0)$ to $(0,1)$.

Given two segments $\mathbf{S}_o$ and $\mathbf{S}_o'$, the comparison relationship between the similarity scores of the two corresponding objects under weighting vector $\mathbf{v}$, i.e., $ST(o,q,\mathbf{w})$ and $ST(o',q,\mathbf{w})$, can be summarized in Theorem 3.2.2.

**Theorem 3.2.2.** Consider a weighting vector $\mathbf{v}$ in a query and two objects’ fixed-score segments $\mathbf{S}_o$ and $\mathbf{S}_o'$ in the “ws–vt” plane (see Fig. 3.2). Let $l_\mathbf{w}$ denote the line of $\mathbf{w}$ and let $p_o$ ($p_o'$) denote the intersections of $l_\mathbf{w}$ and $\mathbf{S}_o$ ($\mathbf{S}_o'$). It holds that $ST(o,q,\mathbf{v})$ > $ST(o',q,\mathbf{v})$ if $|Op_o| < |Op_o'|$ (i.e., $p_o$ is closer to $O$ than $p_o'$).

**Proof.** Let $(ws_o, wt_o)$ and $(ws_o', wt_o')$ denote the coordinates of $p_o$ and $p_o'$, and let $\mathbf{w} = (w_s, w_t)$. According to the property of fixed-score segments, $ws_o \cdot ws_o + wt_o \cdot vt_o = ws_o' \cdot ws_o' + wt_o' \cdot vt_o' = C$. Since $p_o$ and $p_o'$ are located on the line of $\mathbf{w}$, both

$$\frac{ws}{ws_o} = \frac{wt}{wt_o} = \frac{|\mathbf{v}|}{|Op_o|} \quad \text{and} \quad \frac{ws'}{ws_o'} = \frac{wt'}{wt_o} = \frac{|\mathbf{v}|}{|Op_o'|}$$

hold. Hence, $ST(o,q,\mathbf{v}) = ws \cdot ws_o + wt \cdot vt_o = |\mathbf{v}| \cdot (ws_o \cdot ws_o + wt_o \cdot vt_o) = |\mathbf{v}| \cdot C$. Similarly, $ST(o',q,\mathbf{v}) = \frac{|\mathbf{v}}{|Op_o'|} \cdot C$. As we assume $|Op_o| < |Op_o'|$, it holds that $ST(o,q,\mathbf{v}) > ST(o',q,\mathbf{v})$. □

If the condition $|Op_o| < |Op_o'|$ in Theorem 3.2.2 holds, we also say $\mathbf{S}_o$ is closer to the origin than $\mathbf{S}_o'$ in the $\mathbf{w}$ direction. By Theorem 3.2.2, we obtain the following proposition.
Proposition 3.2.3. Let $\vec{w}$ be the weighting vector in a query. For a given object $o$, if in the “ws–wt” plane, there are $k$ objects with fixed-score segments closer to the origin than $S_o$ in the $\vec{w}$ direction, the spatial keyword similarity ranking of $o$ must be $k + 1$.

Proposition 3.2.3 provides a straightforward method to determine the ranking of $o$ for a given query $q$. First, we compute the fixed-score segments of objects in object set $D$. Then we draw the line $l_{\vec{w}}$ and intersect it with all objects’ fixed-score segments. The distances of these intersections to the origin determine the rankings of the objects.

3.2.2 Query Modification and Ranking Updates

Proposition 3.2.3 also implies how we update the ranking of a given object when the weighting vector is modified. Since a weighting vector must correspond to a vector from the origin to a point on the line segment $nw$, the modification can be regarded as a vector rotation in the “ws–wt” plane, in which the end point of the vector must be on $nw$. Take Fig. 3.3 as an example. Here $\vec{w}_0$ is the initial weight, and $\vec{w}_1$ is obtained from $\vec{w}_0$ by increasing $wt$ and decreasing $ws$. In the “ws–wt” plane, we can obtain $\vec{w}_1$ by counterclockwise rotating $\vec{w}_0$ and keeping the endpoint on segment $nw$. Accordingly, the line of weighting vector rotates by the same angle.
The rotation of line \( \vec{l} \) changes its intersection points with the fixed-score segments, which means that the rankings of objects also change. In Fig. 3.3, after \( \vec{l} \) rotates counterclockwise across the intersection point of \( S_o \) and \( S_{o_1} \) (labeled by \( p_1 \)), \( S_{o_1} \) becomes closer to the origin than \( S_o \) in the \( \vec{w} \) direction. Thus \( o \) now ranks lower than \( o_1 \), and its ranking should be decremented by 1. Similarly, as \( \vec{l} \) continues to rotate across the intersection point of \( S_o \) and \( S_{o_2} \) (labeled by \( p_2 \)), \( o \) now ranks higher than \( o_2 \) and its ranking should be incremented by 1. The case is similar for clockwise rotation. That is, after \( \vec{l} \) encounters \( p_3 \), the intersection point of \( S_o \) and \( S_{o_3} \), the ranking of \( o \) should be decremented by 1, while after \( \vec{l} \) encounters \( p_4 \), the ranking should be incremented by 1.

If two objects’ fixed-score segments \( S_o \) and \( S_{o'} \) do not intersect, they will not affect the ranking of each other as the weighting vector rotates. In other words, the ranking updates of \( o \) occur if and only if \( \vec{l} \) encounters the intersections of \( S_o \) and other objects’ fixed-score segments. To formally define this condition, we categorize intersections on \( S_o \) as: promoted points and degraded points.

**Definition 3.2.4. Promoted Points and Degraded Points.** Let \( p \) be an intersection point of \( S_o \) and another object’s fixed-score segment. If, after rotating the weighting vector, \( \vec{l} \) encounters \( p \) and the ranking of \( S_o \) is promoted, we call \( p \) a promoted point; otherwise, if the ranking of \( S_o \) is degraded, \( p \) is called a degraded point.

In the example of Fig. 3.3, \( p_1 \) and \( p_3 \) are degraded points, while \( p_2 \) and \( p_4 \) are promoted points. We observe that all promoted points are contributed by the fixed-score segments of the dominators of \( o \) under \( \vec{w}_0 \). Based on the promoted and degraded points, we can derive an alternative method to compute the ranking of a given \( o \) from any weighting vector. Let the ranking of \( o \) under the original weighting vector \( \vec{w}_0 \) be \( n \), and let the refined weighting vector be \( \vec{w}_1 \). We first compute all promoted and degraded points in \([\vec{l}_{\vec{w}_0}, \vec{l}_{\vec{w}_1}]\). Let the numbers of promoted and degraded points be \( n_p \) and \( n_d \), respectively. Thus, the ranking of \( o \) under weighting vector \( \vec{l}_{\vec{w}_1} \) is \( n - n_p + n_d \). For example, in Fig. 3.3 the ranking of \( o \) under the original
weighting vector $\vec{w}_0$ is 3, and the rankings under $\vec{w}_1$ and $\vec{w}_2$ are 4 and 3, respectively.

The above observations yield the following theorem.

**Theorem 3.2.5.** The weighting vector of the best refined query for a missing object $o$ must be either the original weighting vector or must go through a promoted point of $S_o$.

**Proof.** To prove the theorem by contradiction, let the weighting vector of the best refined query be $\vec{w}_b$ and assume that $\vec{w}_b$ is neither the original weighting vector nor goes through a promoted point of $S_o$. There are then only two cases: There is no promoted point within angle interval $[\vec{w}_0, \vec{w}_b]$; or promoted points exist within this angle interval. In the former case, $R(o, q, \vec{w}_b) \geq R(o, q, \vec{w}_0)$. Since the modification of $\vec{w}_b$ in the weight dimension is larger than that of $\vec{w}_0$, the penalty of $\vec{w}_b$ is higher than that of $\vec{w}_0$, which contradicts the assumption that $\vec{w}_b$ is the best refined weight. In the latter case, let $\vec{w}_b'$ denote the promoted point nearest to $\vec{w}_b$ in $[\vec{w}_0, \vec{w}_b]$. Since there is no promoted point in $[\vec{w}_b, \vec{w}_b']$, $R(o, q, \vec{w}_b) \geq R(o, q, \vec{w}_b')$ holds. On the other hand, since $\vec{w}_b'$ is in $[\vec{w}_0, \vec{w}_b]$, the modification of $w_b$ is larger than that of $w_{b'}$. As such, the penalty of $\vec{w}_b$ is larger than that of $\vec{w}_b'$, which contradicts the assumption that $\vec{w}_b$ is the best refined weight. □

### 3.2.3 Basic Algorithm for Why-Not Spatial Keyword Query

Theorem 3.2.5 suggests a basic query processing algorithm that involves four steps. 1) We compute the ranking of $o$ under the original weighting vector by adapting an existing spatial keyword top-$k$ algorithm (e.g., [14]), get all dominators of $o$ under the original weighting vector, and identify the promoted point produced by each such object. 2) We compute the fixed-score segments of all other incomparable objects of $o$ in $D$ except the dominators under $\vec{w}_0$, and we intersect them with $S_o$. These intersection points are degraded points. An incomparable object $o'$ of $o$ can be obtained by two range queries: $SDist(o', q) < SDist(o, q) \wedge TSim(o', q) < TSim(o, q)$ and $SDist(o', q) > SDist(o, q) \wedge TSim(o', q) > TSim(o, q)$. 3) We compute the rankings under all promoted points that are introduced by these promoted and degraded
points. 4) We compute the penalty values for all promoted points and the original weighting vector, and we select the one with the lowest penalty value.

As an example, let us revisit Fig. 3.3. 1) We compute $R(o, q, \tilde{w}_0)$ (=3) and find all dominators of $o$ under $\tilde{w}_0$ ($\{o_2, o_4\}$). We then compute the fixed-score segments of the dominators ($S_{o_2}$ and $S_{o_4}$) and intersect them with $S_o$ to obtain $p_2$ and $p_4$ as promoted points. 2) We intersect the fixed-score segments of the remaining incomparable objects ($S_{o_1}$ and $S_{o_3}$) with $S_o$, and we identify the intersections $p_1$ and $p_3$ that are degraded points. 3) Using these promoted points and degraded points, we compute the rankings of $o$ under the weighting vectors going through $p_2$ and $p_4$. 4) Since the ranking of both promoted points is 3, which is not higher than that under the original weighting vector, the best refined query is obtained by enlarging $k$ without changing the weighting vector.

### 3.3 Bound and Prune Algorithm

The basic processing algorithm avoids exhaustively enumerating an infinite number of candidate weighting vectors by searching only those going through promoted points. However, it still requires the use of two range queries to find degraded points. Since the number of degraded points can be large, the algorithm still has high computation cost. We proceed to present an optimized algorithm that uses a new index structure, the BIR-tree (Bounded IR-tree), to prune unnecessary accesses to objects and promoted points, thus improving efficiency. The essence of the BIR-tree is that by accessing high-level nodes, we can estimate the number of degraded points in some range without actually accessing them.

Section 3.3.1 introduces the BIR-tree, and Section 3.3.2 gives an overview of our optimized algorithm and discusses how to use upper and lower bounds to estimate the number of degraded points. Then Section 3.3.3 describes how to compute the upper and lower bounds using a BIR-tree.
### 3.3.1 A Hybrid Index: BIR-Tree

The BIR-tree is a hybrid data structure that indexes both the spatial and textual attributes of spatial web objects. It is a variant of the IR-tree [14] and Fig. 3.4 illustrates an example. A leaf node contains a number of entries of the form of \((o, mbr, di)\), where \(o\) represents an object, \(mbr\) is the minimum bounding rectangle (MBR) of the object, and \(di\) is a document identifier of the object. A non-leaf node contains a number of entries of the form \((cp, mbr, di)\), where \(cp\) is a pointer to a child node, \(mbr\) is the MBR of the child node, and \(di\) is an identifier of a pseudo document that represents all documents in the child node’s subtree. In addition, each node in the BIR-tree stores a \(cnt\) value, which is the number of objects in the node’s subtree, and a pointer (shown as an arrow in the figure) to an inverted file for these objects. In the inverted file, we maintain for each term \(t\) two bounds on its weight. \(\hat{w}(t)\) is the maximum weight of \(t\) and is used to estimate the upper bound of textual similarity for the objects in the entry’s subtree, while \(\check{w}(t)\) is the minimum weight and is used for the estimation of the lower bound of textual similarity. For example, in \(R_3\)’s inverted file, the term “Chinese” has its \(\hat{w}(t) = 5\) and \(\check{w}(t) = 0\) for estimating the bounds on textual similarities for objects in \(R_1\). For clarity of presentation, here we use the keyword frequency to represent a weight.\(^1\)

![Figure 3.4: Structure of a BIR-Tree](image-url)
Algorithm 1 Optimized Why-Not Spatial Keyword Query Processing Algorithm

by Estimation

INPUT: BIR-tree $\mathcal{T}$, original query $q = (loc, doc, k, w_0)$, missing object $o$

OUTPUT: Best refined query $q_b = (loc, doc, k_b, w_b)$

1: determine $R(o, q, \hat{w}_0)$ and compute $o$'s dominators under the original query
2: $Pro \leftarrow \emptyset$ // set of promoted points
3: for each dominator $o'$ of $o$ under the original query
4: intersect $S_{o'}$ with $S_o$
5: if there is an intersection of $S_o$ and $S_{o'}'$ then
6: $Pro \leftarrow Pro \cup \{p(S_o, S_{o'})\}$
7: $TH \leftarrow$ penalty of $w_0$ // the threshold recording the minimum penalty
8: for each element $e$ in $Pro$
9: $PP(e) \leftarrow \#$ promoted points between $e$ and $p(l_{w_0}, S_o)$
10: $\hat{R}(e) \leftarrow R(o, q, \hat{w}_0) - PP(e) + DP(T.root, e)$ // Section 3.3.3
11: $\hat{R}(e) \leftarrow R(o, q, \hat{w}_0) - PP(e) + DP(T.root, e)$ // Section 3.3.3
12: use $\hat{R}(e)$ and $\hat{R}(e)$ to compute the lower bounds and upper bounds of penalty of $e$, i.e., $\hat{p}_n(e)$ and $\hat{p}_n(e)$
13: if $\hat{p}_n(e) > TH$ then prune $e$ from $Pro$
14: $Q \leftarrow$ an empty queue
15: insert $T.root$ into $Q$
16: while $Q$ is not empty do
17: $N \leftarrow$ Dequeue($Q$)
18: if $N$ has no degraded part or no existing promoted point on $N$'s degraded and promoted part
19: then continue
20: for each existing promoted point $e$ on $N$'s degraded part
21: $DP' \leftarrow 0$
22: $\hat{DP}' \leftarrow 0$
23: for each child $c$ of $N$
24: $DP' \leftarrow DP' + DP(c, e)$
25: $\hat{DP}' \leftarrow \hat{DP}' + DP(c, e)$
26: $\hat{R}(e) \leftarrow \hat{R}(e) - (DP(N, e) - \hat{DP}')$
27: $\hat{R}(e) \leftarrow \hat{R}(e) + (\hat{DP}' - DP(N, e))$
28: use $\hat{R}(e)$ and $\hat{R}(e)$ to compute $\hat{p}_n(e)$ and $\hat{p}_n(e)$
29: if $\hat{p}_n(e) < TH$ then $TH \leftarrow \hat{p}_n(e)$
30: for each existing promoted point $e$ in $N$'s degraded part
31: if $\hat{p}_n(e) > TH$ then prune $e$ from $Pro$
32: insert each child $c$ of $N$ into $Q$
33: for each remaining promoted point $e$
34: $R(o, q, \bar{e}) \leftarrow \hat{R}(e)$
35: compute $p_n(e)$ by $R(o, q, \bar{e})$
36: $b \leftarrow$ the remaining promoted point with the minimum penalty
37: if $p_n(b) < \text{penalty of } w_0$ then return $(loc, doc, R(o, q, \bar{w}_0), \bar{w}_0)$
38: else return $(loc, doc, R(o, q, \bar{w}_0), \bar{w}_0)$
3.3.2 Optimized Query Processing by Estimated Bounds

The optimized algorithm estimates the number of degraded points instead of actually accessing them. Algorithm 1 shows the pseudo-code of this algorithm. According to Theorem 3.2.5, the final best refined query must go through a promoted point or be the initial weighting vector. As such, our aim is to efficiently compute the ranking of each under each weighting vector going through a promoted point. Let $PP(e)$ and $DP(e)$ denote the number of promoted points and the number of degraded points located between a promoted point $e$ and $p(l_{w_0}, S_o)$. For points located exactly in the same place, the number is counted multiple times. As discussed in Section 3.2.2, $R(o, q, \bar{w}_e) = R(o, q, \bar{w}_0) - PP(e) + DP(e)$. We first compute all promoted points based on the dominators under the initial weighting vector (lines 1–6). We maintain $TH$, the current minimal upper bound on the penalty value, and we initialize it to the penalty value of the initial weighting vector (line 7). Next, we traverse the BIR-tree by starting from the root. By sorting the promoted points, $PP(e)$ of each promoted point $e$ can be obtained directly (line 9). The next task is to compute $DP(e)$ of each $e$. We estimate their upper bound $\hat{DP}(N, e)$ and lower bound $\hat{DP}(N, e)$ (Section 3.3.3 provides the details). By these bounds, we get the upper bound $\hat{p}_n(e)$ and lower bound $\hat{p}_n(e)$ on the penalty value (lines 10–12). If the lower bound $\hat{p}_n(e)$ of a promoted point exceeds $TH$, this promoted point cannot be the best refined query and is pruned from the candidate list (lines 13). After that, we recursively access the BIR-tree nodes (lines 14–32). If a node cannot tighten the bounds for any promoted point, we prune it (lines 18–19). Similar to the case of the root, we may prune a promoted point if its lower bound $\hat{p}_n(e)$ on the penalty value exceeds $TH$ (lines 20–31). By pruning nodes and promoted points, we obtain the promoted point with the minimum penalty value (lines 33–36). If this penalty value is less than the penalty of the initial weighting vector, the corresponding promoted point is returned as the answer (line 37). Otherwise, the initial weighting vector is returned.

The literature (e.g., [30]) explains how to compute the weight of a keyword from keyword frequencies.
Given a BIR-tree node $N$ and a promoted point $e$, bound estimation is accomplished as follows. Let $DP(N, e)$ denote the number of degraded points produced by the objects under node $N$ and located between $e$ and $p(l_{w0}, S_o)$. Then $DP(e)$ can be expressed as $\sum_{N \in S} DP(N, e)$, where $S$ is a set of disjoint nodes fully covering all objects in $D$. The upper and lower bounds of $DP(e)$ are initialized to $\hat{DP}(T.root, e)$ and $\tilde{DP}(T.root, e)$, respectively. As we traverse the BIR-tree downwards and access more nodes, the bounds will gradually be tightened by $\hat{DP}(N, e) - \sum \hat{DP}(c, e)$ and $\tilde{DP}(c, e) - \sum \tilde{DP}(N, e)$, where $c$ denotes a child of $N$ (Lines 21–27). In Section 3.3.3, we prove that a promoted point can tighten the bounds only if it is located in the degraded part of the node.

### 3.3.3 Derivation of $\hat{DP}(N, e)$ and $\tilde{DP}(N, e)$

As mentioned, the estimations of $\hat{DP}(N, e)$ and $\tilde{DP}(N, e)$ are key points in the optimized algorithm. Here we first identify the properties of promoted and degraded points and the BIR-tree, then present derivations of $\hat{DP}(N, e)$ and $\tilde{DP}(N, e)$.

#### Properties of Promoted and Degraded Points

Intuitively, whether the fixed-score segment $S_o$ of an object $o$ produces a promoted or degraded point on $S_o$ depends on two relations: the relation between $S_o$ and $S_o'$ and the relation between the intersection point and the initial weighting vector.

**Definition 3.3.1. Relations of Fixed-Score Segments.** Two objects’ fixed-score segments $S_o$ and $S_{o'}$ have four possible relations, determined by the relations of their intercepts in the $ws$ and $wt$ dimensions ($\frac{C}{v_{s0}}$ vs. $\frac{C}{v_{s{'}0}}$ and $\frac{C}{v_{t0}}$ vs. $\frac{C}{v_{t{'}0}}$):

1. If $\frac{C}{v_{t0}} < \frac{C}{v_{t{'}0}}$ and $\frac{C}{v_{s0}} < \frac{C}{v_{s{'}0}}$, $o'$ always ranks higher than $o$ and $o'$ is a dominator of $o$ under all candidate weighting vectors (denoted by $Dom(o)$);  

2. If $\frac{C}{v_{t0}} \geq \frac{C}{v_{t{'}0}}$ and $\frac{C}{v_{s0}} \geq \frac{C}{v_{s{'}0}}$, $o'$ never ranks higher than $o$ and $o'$ is a dominatee under all candidate weighting vectors (denoted by $DomBy(o)$);
3. If \( \frac{C_{vt,o}}{vt_o} \geq \frac{C_{vt,o'}}{vt_{o'}} \) and \( \frac{C_{vs,o'}}{vs_{o'}} < \frac{C_{vs,o}}{vs_o} \), \( S_o \) and \( S_{o'} \) have an intersection point, and \( o' \) is called a steeper object of \( o \) (denoted by \( \text{Stp}(o) \));

4. If \( \frac{C_{vt,o}}{vt_o} < \frac{C_{vt,o'}}{vt_{o'}} \) and \( \frac{C_{vs,o'}}{vs_{o'}} \geq \frac{C_{vs,o}}{vs_o} \), \( S_o \) and \( S_{o'} \) also have an intersection point, and \( o' \) is called a gentler object of \( o \) (denoted by \( \text{Gent}(o) \)).

Take Fig. 3.3 as an example. Here, \( o_3 \) is in \( \text{DomBy}(o_2) \), while \( o_2 \) is in \( \text{Dom}(o_3) \). Both \( o_3 \) and \( o_2 \) are in \( \text{Stp}(o) \), while both \( o_4 \) and \( o_1 \) are in \( \text{Gent}(o) \). As mentioned above, only the fixed-score segments of \( \text{Stp}(o) \) and \( \text{Gent}(o) \) intersect with \( S_o \).

The following lemma shows how the second relation, i.e., the relation between the intersection point and the initial weighting vector, determines the promoted and degraded points.

**Lemma 3.3.2.** A fixed-score segment \( S_{o'} \) produces a promoted point on \( S_o \) if and only if: either 1) \( o' \) is a steeper object of \( o \) and \( p(S_{o'}, S_o) \) is to the left of the intersection \( p(l_{\tilde{w}_0}, S_o) \); or 2) \( o' \) is a gentler object of \( o \) and \( p(S_{o'}, S_o) \) is to the right of the intersection \( p(l_{\tilde{w}_0}, S_o) \). Similarly, \( S_{o'} \) produces a degraded point on \( S_o \) if and only if: either 1) \( o' \) is a steeper object of \( o \) and \( p(S_{o'}, S_o) \) is to the right of the intersection \( p(l_{\tilde{w}_0}, S_o) \); or 2) \( o' \) is a gentler object of \( o \) and \( p(S_{o'}, S_o) \) is to the left of the intersection \( p(l_{\tilde{w}_0}, S_o) \).

The proof is straightforward and thus omitted. In the example of Fig. 3.3, if the initial weighting vector is \( \tilde{w}_0 \), \( S_{o_1} \) and \( S_{o_3} \) produce promoted points on \( S_o \), while \( S_{o_2} \) and \( S_{o_4} \) produce degraded points.

Properties of the BIR-Tree

As the name suggests, the information stored in each node \( N \) of the BIR-tree summarizes the spatial and textual similarity bounds between a query \( q \) and the objects in its subtree. With the spatial bounds, we can compute the maximum distance (denoted by \( \hat{SDist}(N,q) \)) and minimum distance (denoted by \( \check{SDist}(N,q) \)) from the objects in \( N \) to the location of a query \( q \). With the similarity bounds of each keyword, we can compute the upper bound (\( \hat{TSim}(N,q) \)) and lower
Figure 3.5: An Example of the Boundary Region of a Node $N$

bound ($\hat{T}Sim(N, q)$) of textual similarities between the objects in $N$ and a query $q$. Let $\hat{v}s_N = 1 - \hat{SDist}(N, q)$, $\hat{v}s_N = 1 - \hat{SDist}(N, q)$, $\hat{v}t_N = \hat{T}Sim(N, q)$, and $\hat{v}t_N = \hat{T}Sim(N, q)$. We can draw two boundary segments $\hat{S}_N$ and $\hat{S}_N$ for each node $N$ and $q$, where $\hat{S}_N$ is the segment with ends $(\frac{C}{\hat{v}s_N}, 0)$ and $(0, \frac{C}{\hat{v}t_N})$, and $\hat{S}_N$ is the segment with ends $(\frac{C}{\hat{v}s_N}, 0)$ and $(0, \frac{C}{\hat{v}t_N})$. The region enclosed by the $vt$-axis, $\hat{S}_N$, the $vs$-axis, and $\hat{S}_N$ is called the boundary region of $N$ and is denoted by $BA(N)$ (see the shaded region in Fig. 3.5). The following theorem establishes a desirable property of $BA(N)$.

**Theorem 3.3.3.** For any node $N$ in the BIR-tree, the fixed-score segments of all objects in $N$ must be covered by $BA(N)$.

**Proof.** Fig. 3.5 illustrates the theorem. Since $\hat{SDist}(N, q)$ is the lower bound of the distance from the objects in $N$ to $q$, for any object $o$ in $N$, $SDist(o, q) \geq \hat{SDist}(N, q)$. So, $\hat{v}s_o \leq \hat{v}s_N$, from which $\frac{C}{\hat{v}s_o} \geq \frac{C}{\hat{v}s_N}$ follows. Similarly, $\frac{C}{\hat{v}s_o} \leq \frac{C}{\hat{v}s_N}$.

This means that one end of $S_o$ is on the segment from $(0, \frac{C}{\hat{v}s_N})$ to $(0, \frac{C}{\hat{v}s_N})$, which is an edge of $BA(N)$. Likewise, another end of $S_o$ on the $wt$-axis is also on the edge of $BA(N)$ on the $wt$-axis. Since $BA(N)$ is convex, the full segment $S_o$ is covered by $BA(N)$. $\Box$

Next, the relation between a node $N$ and an object not in $N$ can be divided into nine cases, according to the relation between the intercepts of $S_o$ and the edges of
Table 3.3: Relations between Node $N$ and Object $o$

<table>
<thead>
<tr>
<th>$S_o^s$</th>
<th>$(\hat{S}_N^s, \infty)$</th>
<th>$[\hat{S}_N^s, \hat{S}_N^t]$</th>
<th>$(0, \hat{S}_N^t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\hat{S}_N^s, \infty)$</td>
<td>▲$^1$</td>
<td>▲$\odot^2$</td>
<td>▲$^3$</td>
</tr>
<tr>
<td>$[\hat{S}_N^s, \hat{S}_N^t]$</td>
<td>▼▼$^4$</td>
<td>▼▼▲$^5$</td>
<td>▼▲$^6$</td>
</tr>
<tr>
<td>$(0, \hat{S}_N^t)$</td>
<td>▼$^7$</td>
<td>▼▲$^8$</td>
<td>▼$^9$</td>
</tr>
</tbody>
</table>

$\odot = Stp(o)$, ▲ = Gent(o), ▲ = Dom(o), ▲ = DomBy(o)

$BA(N)$ on the $ws$-axis and the $wt$-axis. For ease of presentation, let $S_o^s$ ($S_o^t$) denote the intersection point between segment $S_o$ and the $ws$-axis ($wt$-axis).

Combining the properties of BIR-tree and the promoted and degraded points, we can also identify the relations between the objects in $N$ and $S_o$ as follows. If both $S_o^s \in (\hat{S}_N^s, \infty)$ and $S_o^t \in (0, \hat{S}_N^t)$ hold, all objects in $N$ are steeper objects of $o$ (like $N$ and $o_1$ in Fig. 3.5); likewise, if both $S_o^t \in (\hat{S}_N^t, \infty)$ and $S_o^s \in (0, \hat{S}_N^s)$ hold, all objects in $N$ are gentler objects of $o$ (like $N$ and $o_2$ in Fig. 3.5). Other relations between $N$ and $o$ are summarized in Table 3.3, and we number the cases through 1 to 9 for latter reference. The column and row headers indicate the intervals that $S_o^s$ and $S_o^t$ belong to. The symbols used are explained in the last row. For example, ▼ means that the objects in $N$ belong $Stp(o)$. If $S_o^s \in [\hat{S}_N^s, \hat{S}_N^t]$ and $S_o^t \in (\hat{S}_N^t, \infty)$ hold, symbols ▲$\odot$ apply meaning that the objects in $N$ belong to $Dom(o)$ or to $Gent(o)$.

**Estimation of $\hat{D}P(N,e)$ and $\bar{D}P(N,e)$**

The bounds on $DP(N,e)$ depend on two factors: the characteristic of $N$ and the location of the promoted point $e$. The former determines the total number of degraded points in $N$, while the latter determines the number of degraded points among them that are located between $e$ and $p(l_{\bar{w}_o}, S_o)$, i.e., $DP(N,e)$. We proceed to analyze these two factors.

As discussed earlier in this section, a node not containing $o$ contains 4 kinds of objects, i.e., objects in $Dom(o)$, $DomBy(o)$, $Gent(o)$, or $Stp(o)$. Formally,
\[ N \text{.cnt} = |N \text{.Dom}(o)| + |N \text{.DomBy}(o)| + |N \text{.Gent}(o)| + |N \text{.Stp}(o)|. \]  
(3.3.4)

Since the fixed-score segments of \( \text{Stp}(o) \) and \( \text{Gent}(o) \) must intersect with that of the missing object \( o \) and produce promoted or degraded points on \( S_o \), we can deduce that:

\[ |N \text{.Gent}(o)| + |N \text{.Stp}(o)| = PP(N) + DP(N), \]  
(3.3.5)

where \( PP(N) \) (\( DP(N) \)) denotes the number of promoted (degraded) points produced by the objects in the subtree of entry \( N \) and located on the segment \( S_o \).

Substituting Eqn. 3.3.5 into Eqn. 3.3.4, the number of degraded points on \( S_o \) produced by the objects in \( N \)’s subtree can be expressed as:

\[ DP(N) = N \text{.cnt} - |N \text{.Dom}(o)| - |N \text{.DomBy}(o)| - PP(N). \]  
(3.3.6)

Based on these equations and the nine different relations between a node and the missing object’s segment in Table 3.3, we estimate bounds on \( DP(N,e) \) for each relation as follows.

For relation 1 in Table 3.3, all objects are in \( \text{Dom}(o) \), i.e., \( N \text{.Gent}(o) = \emptyset \) and \( N \text{.Stp}(o) = \emptyset \). According to Eqn. 3.3.5, we can deduce that \( DP(N) = \emptyset \). That is to
say, this kind of node can never produce degraded points on $S_o$; thus, they can be pruned. This also applies to relation 9, where only $DomBy(o)$ is non-empty.

For relation 5, a node covers all of segment $S_o$, and may contain all four kinds of objects. Since we can only get the information about $Dom(o)$, the promoted points in $S_o$ (denoted by $PP(S_o)$), and part of the degraded points, we cannot estimate the cardinality of $DomBy(o)$. That is to say, in Eqn. 3.3.6, we only know $|N.Dom(o)| \leq |Dom(o)|$, $PP(N) \leq PP(S_o)$. As such, we can only conclude that $DP(N) \in [0, N.cnt]$, and therefore wherever $e$ is located, we set the bounds of $DP(N, e)$ to $[0, N.cnt]$.

We now focus on the remaining cases. For relations 2 and 4, $N.DomBy(o) = \emptyset$. According to Eqn. 3.3.6, the number of degraded points produced by the objects in such a node $N$ is:

$$DP(N) = N.cnt - |N.Dom(o)| - PP(N).$$  \hspace{1cm} (3.3.7)

In one extreme, we set the cardinality of $Dom(o)$ (denoted by $|Dom(o)|$) as the upper bound of $|N.Dom(o)|$, and the total number of promoted points on the promoted part (denoted by $PP(S_{o, N}^+)$ and to be defined in Definition 3.3.4) as the upper bound of $PP(N)$. According to Eqn. 3.3.7, the lower bound of $DP(N)$ can be estimated as:

$$\Hat{DP}(N) = \max\{0, N.cnt - PP(S_{o, N}^+) - |Dom(o)|\}. \hspace{1cm} (3.3.8)$$

In the other extreme, we assign 0 as the lower bound of $PP(N)$ and $|N.Dom(o)|$. By Eqn. 3.3.7, we can derive the upper bound of $DP(N)$ by the following equation:

$$\Hat{DP}(N) = N.cnt. \hspace{1cm} (3.3.9)$$

For relations 3 and 7, nodes only contain objects that belong to $Gent(o)$ or $Stp(o)$, i.e., $N.Dom(o) = N.DomBy(o) = \emptyset$. According to Eqn. 3.3.6, $DP(N) = N.cnt - PP(N)$. As $PP(N) \in [0, PP(S_{o, N}^+)]$, the range of $DP(N)$ is $[\max\{0, N.cnt - PP(S_{o, N}^+), N.cnt\}]$.

For the remaining two relations 6 and 8, $N.Dom(o) = \emptyset$. According to Eqn. 3.3.6,
DP(N) = N.cnt − |N.DomBy(o)| − PP(N),

(3.3.10)

where we only know \( PP(N) \in [0, PP(S_{o,N}^+)] \), but not \( |D.DomBy(o)| \). We therefore set \( DP(N) \) to be \([0, N.cnt]\).

Now we take the next step to derive \( DP(N, e) \) based on the location of \( e \). We first introduce three concepts, promoted part, degraded part, and influenced part.

Relations 1 and 9 never produce degraded points, and for relation 5, we have shown that wherever \( e \) locates, \( DP(N, e) \in [0, N.cnt] \). This leaves us with 6 relations, some of which are shown in Fig. 3.6. For example, Fig. 3.6(a) corresponds to relation 4, i.e., \( S_o^s \in (\hat{S}_N, \infty) \) and \( S_o' \in [\hat{S}_N, \hat{S}_N] \). According to Table 3.3, each object \( o' \) in \( N \) must be in \( Dom(o) \) or \( Stp(o) \). If \( o' \) is in \( Stp(o) \), \( S_{o'} \) must intersect with \( S_o \) on the segment \( \overline{p_lp_r} \), where \( \overline{p_lp_r} \) is the part of \( S_o \) covered by \( BA(N) \) and \( p_l \) \( (p_r) \) is the left (right) most end of this subsegment. For simplicity, we denote \( p(l_{o0}, S_o) \) as \( p_0 \). According to Lemma 3.3.2, if \( p(S_o, S_{o'}) \) lies to the left of \( p_0 \) (i.e., on the subsegment \( \overline{p_lp_0} \)), it must be a promoted point; otherwise, if \( p(S_o, S_{o'}) \) lies on the subsegment \( \overline{p_0p_r} \), it must be a degraded point. We call \( \overline{p_lp_0} \) the promoted part of \( S_o \) and \( \overline{p_0p_r} \) the degraded part of \( S_o \).

**Definition 3.3.4. Promoted Part and Degraded Part.** For a given object \( o \), node \( N \) in relations 2, 3, 4, 6, 7, 8, the subsegment of \( S_o \) covered by \( BA(N) \) can be divided into two disjoint parts: a promoted part (denoted by \( S_{o,N}^+ \)) and a degraded part (denoted by \( S_{o,N}^- \)). For any object \( o' \) in \( N \), if \( p(S_o, S_{o'}) \) is on the promoted part, it must be a promoted point; and if \( p(S_o, S_{o'}) \) is on the degraded part, it must be a degraded point.

Fig. 3.6 illustrates the promoted and degraded parts, which are denoted by dotted and dashed segments, respectively. If \( S_o^s \in (\hat{S}_N, \infty) \), either \( S_o' \in [\hat{S}_N, \hat{S}_N] \) (see Fig. 3.6(a)) or \( S_o' \in (\hat{S}_N, \infty) \) (see Fig. 3.6(b)), the promoted part is to the left of \( p_0 \), and the degraded part is to the right. Conversely, if \( S_o^s \in (\hat{S}_N, \infty) \), either \( S_o' \in [\hat{S}_N, \hat{S}_N] \) (see Fig. 3.6(c)) or \( S_o' \in (\hat{S}_N, \infty) \) (see Fig. 3.6(d)), the promoted part is to the right, and the degraded part is to the left. Note that there will be
no promoted (see Fig. 3.6(d)) or degraded part (see Fig. 3.6(c)) if the part of $S_o$
covered by $BA(N)$ lies totally to the right or left of $p_0$.

The influenced part denotes the part of $S_o$ on the same side as the degraded part, but outside $BA(N)$. In Fig. 3.6, the influenced part is denoted by a bold line. Note that if there is no degraded part, there will be no influenced part, as shown in Fig. 3.6(c). And for relations 6 and 8, there will be no influenced part, either.

As such, the location of a promoted point $e$ can fall into one of four parts: 1) the influenced part; 2) the degraded part; 3) the promoted part; or 4) the remaining part. For case 1), if $e$ is located on the influenced part of $N$, $DP(N, e) = DP(N)$ because all degraded points produced by the objects in $N$ are located in the interval $(p_0, e)$. According to the previous estimation of $DP(N)$, we have:

$$DP(N, e) = DP(N) = \begin{cases} 
\max\{0, N\text{.cnt} - PP(S_{o,N}^-) - |\text{Dom}(o)|\} & \text{for 2 and 4} \\
\max\{0, N\text{.cnt} - PP(S_{o,N}^+)\} & \text{for 3 and 7}
\end{cases}$$

$$DP(N, e) = \hat{DP}(N) = N\text{.cnt}.$$

For case 2), if $e$ is located on the degraded part of $N$, since we do not know the objects in $N$, an object in $N$ may intersect with $S_o$ either inside or outside the interval $(e, p_0)$. In extreme cases, all or no objects in $N$ are degraded points in the interval $(e, p_0)$. Hence, we set the bounds on $DP(N, e)$ to be $[0, N\text{.cnt}]$, which is looser than the bounds in case 1).

For cases 3) and 4), there is no degraded point in the interval $(e, p_0)$. Thus, the bounds on $DP(N, e)$ are set to $[0, 0]$.

Since the degraded part of a parent node $N$ must contain those of its child nodes, if a promoted point $e$ lies on such a part, it can be located on the degraded or influenced part of its child nodes. Only in the latter case can the bounds of $DP(N, e)$ be tightened because the bounds of case 1) are tighter than those of case 2). As such, in the optimized query processing algorithm, we only need to observe the promoted points on the degraded part of an accessed node, and we then gradually tighten the bounds as we access child nodes (Lines 20–27 in Algorithm 1).
3.4 Handling Multiple Missing Objects

We proceed to extend the algorithms to deal with queries with multiple missing objects. Recall that the goal of a why-not spatial keyword query is to refine the query so that all missing objects are in the result. To achieve when there are multiple missing objects, we first identify the lowest-ranked missing object under arbitrary weighting vectors. To do so, we project the missing objects to the two-dimensional plane. Fig. 3.7 shows an example with four missing objects $o_1, o_2, o_3$, and $o_4$ with fixed-score segments $S_{o_1}, S_{o_2}, S_{o_3},$ and $S_{o_4}$. If the weighting vector is $\tilde{w}_1$, $o_1$ is the lowest-ranked missing object because the intersection of $S_{o_1}$ and the weighting vector line $l_{\tilde{w}_1}$ is the farthest from the origin. Similarly, if $\tilde{w}_2$ is the weighing vector, $o_2$ is the lowest-ranked missing object. Given an arbitrary weighting vector $\tilde{w}$, we define the intersection of the line $l_{\tilde{w}}$ and the farthest fixed-score segment as a working point.

The working parts that correspond to a missing object constitute a sub-segment, which we call a working part (e.g., the bold segments for $o_1, o_2$, and $o_3$ in Fig. 3.7). The promoted points in the working parts are called promoted working points. The following theorem shows an important property of the promoted working points.

**Theorem 3.4.1.** Given a set of missing objects, the weighting vector of the best refined query must be either the initial weighting vector or must go through a promoted working point.
Algorithm 2 Identification of Working Parts

INPUT: Missing object set $M$

OUTPUT: Working parts of missing objects $WP$

1: $WP \leftarrow \emptyset$
2: intersect the fixed-score segments of the objects in $M$ with the $wt$ and $ws$ axes
3: $IS \leftarrow$ the $ws$-intercept set (in ascending order)
4: denote the $i$th element in $IS$ by $is_i$
5: denote the fixed-score segment producing the intercept $is_i$ by $S(is_i)$
6: $n \leftarrow |M|$
7: $sp \leftarrow is_n$
8: while true do
9:   for $i$ from 1 to $n-1$ do
10:      intersect $S(is_i)$ with $S(is_n)$
11:     if there is no intersection then
12:        $S(is_n)^\perp \leftarrow$ the intersection of $S(is_n)$ and the $wt$-axis
13:        insert sub-segment $(sp, S(is_n)^\perp)$ into $WP$
14:        break
15:     else
16:        $pc \leftarrow$ the intersection nearest to $sp$
17:        $S(is_c) \leftarrow$ the object segment producing $pc$
18:        insert sub-segment $(sp, pc)$ into $WP$
19:        $sp \leftarrow pc$
20:        $n \leftarrow c$
21:    return $WP$

Proof. The proof is similar to that of Theorem 3.2.5. □

Based on this property, we first identify the working parts in Section 3.4.1, and then we describe how to use the working parts to process why-not spatial keyword queries with multiple missing objects in Section 3.4.2.

3.4.1 Identifying Working Parts

Algorithm 2 computes the working parts. First, we sort the fixed-score segments of the missing objects in ascending order of their intercepts with the $ws$ axis (Lines 2–3). Obviously, if the weighting vector follows the $ws$ axis, the fixed-score segment
with the largest $ws$-intercept (denoted by $is_n$) is the farthest one. Thus, we initialize this intercept as the starting point $sp$ of the working part for this fixed-score segment (Line 7). Next, we find the intersections between this fixed-score segment and all other fixed-score segments with smaller $ws$-intercepts (Lines 9–10). The nearest intersection point $p_c$ is regarded as the ending point of the working part. The working part $(sp, p_c)$ is then inserted into the result set $WP$ (Lines 16–18). The fixed-score segment producing $p_c$ is denoted by $S(is_c)$, where $is_c$ is the $ws$-intercept of $S(is_c)$. Since $is_c$ is smaller than $is_n$, $S(is_c)$ is steeper than $S(is_n)$. Therefore, if the weighting vector rotates counterclockwise across $p_c$, $S(is_c)$ will be farther farthest from the origin than $S(is_n)$, so it becomes the farthest fixed-score segment. $p_c$ becomes the starting point of the working part for $S(is_c)$. Then we set $n$ to $c$ and repeat the above operations to identify the remaining working parts. The procedure terminates when no intersection exists between $S(is_n)$ and the other fixed-score segments with smaller $ws$-axis intercepts (Line 11). In that case, there is no farther fixed-score segment if the weighting vector rotates from $sp$ to the $wt$-axis. Hence, the sub-segment $(sp, S(is_n)^t)$ is the last identified working part, and it is inserted into the result set, where $S(is_n)^t$ is the intersection of $S(is_n)$ and the $wt$-axis. The time complexity of Algorithm 2 is $O(n^2)$, where $n$ is the number of missing objects.

Fig. 3.7 exemplifies the algorithm. Initially, the fixed-score segments are sorted as $\{S_{o_1}, S_{o_2}, S_{o_4}, S_{o_3}\}$ according to their $ws$-axis intercepts, and $S_{o_1}$ is the initial, farthest fixed-score segment. We intersect $S_{o_3}$ with $S_{o_1}, S_{o_2}$, and $S_{o_4}$; the nearest intersection point is $p(S_{o_2}, S_{o_3})$. Consequently, the working part for $S_{o_3}$ is the sub-segment $(p(S_{o_3}, ws), p(S_{o_3}, S_{o_2}))$. $is_n$ becomes $S_{o_2}$, and $p(S_{o_2}, S_{o_3})$ is the starting point of its working part. Since the $ws$-intercept of $S_{o_4}$ exceeds that of $S_{o_2}$ and $p(S_{o_4}, S_{o_3})$ is farther from $p(S_{o_3}, ws)$ than $p(S_{o_2}, S_{o_4})$, there is no working part for $S_{o_4}$. Now $S_{o_1}$ is the only fixed-score segment with smaller $ws$-intercept. Thus, we intersect $S_{o_1}$ with $S_{o_2}$ and get an intersection point $p(S_{o_1}, S_{o_2})$ that serves as the ending point of the working part for $S_{o_2}$. Finally, since there is no fixed-score segment with $ws$-intercept smaller than $S_{o_1}$, the sub-segment $(p(S_{o_1}, S_{o_2}), S_{o_1}^t)$ is
the working part for $S_{oi}$.

3.4.2 Why-Not Spatial Keyword Query Processing for Multiple Missing Objects

Following Theorem 3.4.1, query processing for multiple missing objects can be divided into three steps: 1) Finding all promoted working points; 2) computing the rankings of the farthest fixed-score segments under the weighting vectors that go through the promoted working points; 3) computing the penalty values under the above weighting vectors and selecting the best refined query.

The first step starts right after identifying the working parts. We call those missing objects that contain working parts working missing objects. We first compute the dominators of the working missing objects under the initial weighting vector. Then we identify the dominators that will produce promoted points on the working parts of the working missing objects. Next, we intersect the fixed-score segments of these dominators with those of the corresponding working missing objects. If the intersections are located on the working parts, they become the promoted working points.

The second step is done by slightly modifying the basic algorithm for a single missing object. Specifically, we intersect all incomparable objects in $D$ with each working missing object and compute the number of degraded points between each promoted working point and the initial weighting vector. Using these degraded points and previously computed promoted points, the ranking of each weighting vector can be obtained by the algorithm described in Section 3.2.2. This step can also be accomplished using a modification of the optimized algorithm. After accessing a node $N$, for each promoted working point, we compute the bounds of degraded points caused by the objects under $N$ and the corresponding working missing objects. With these bounds, we can also estimate the total bound of the ranking for each promoted working point.

After obtaining these rankings, the penalty values can be derived by Eqn. 3.1.3
in the third step.

## 3.5 Empirical Study

### 3.5.1 Experimental Setup

**Algorithms for Comparison**

We implemented four algorithms for study. The first is the baseline algorithm (Baseline, Section 3.1.3), the second is the basic algorithm (BS), the third is the optimized algorithm, which employs the bound and prune strategy (BP), and the fourth is an algorithm based on a principle developed by He and Lo [19] (TM). This last algorithm computes the rankings of the missing objects under all candidate weighting vectors. As stated by Theorem 3.2.5, the candidate weighting vectors are either the initial weight vector or go through a promoted point of $S_o$. The ranking of a missing object is derived by slightly modifying a typical spatial keyword top-$k$ query processing algorithm (i.e., [14])—instead of finding $k$ objects with the largest scores, the algorithm stops when the missing object is popped from the max heap that sorts the accessed IR-tree nodes and objects. For fairness, we also implemented two optimizations in TM. First, deriving the ranking of the missing object stops earlier, namely when the number of popped objects leads to a penalty value exceeding the current minimum penalty. Second, we adopt the properties in Section 3.3 to estimate the upper bound of the ranking under some weighting vector, i.e., when $R(o, q, \tilde{w}_1)$ is being computed with another weighting vector $\tilde{w}_2$, $R(o, q, \tilde{w}_1) - R(o, q, \tilde{w}_2)$ will not exceed the number of promoted points between them.

**Datasets**

We use the real datasets EURO and GN in the experiments. Both are commonly used in the spatial keyword query research [4, 5, 29, 39]. Each dataset contains

---

a number of objects with location coordinates and a set of keywords. EURO is a dataset of points of interest such as ATMs, hotels and stores in Europe; and GN is a set of geographic objects obtained from the US Board on Geographic Names. Their characteristics are given in Table 4.2.

<table>
<thead>
<tr>
<th>Table 3.4: Dataset Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>total # objects</td>
</tr>
<tr>
<td># distinct words</td>
</tr>
<tr>
<td>Avg. # words per object</td>
</tr>
</tbody>
</table>

System Setup and Metrics

Our experiments were conducted on a PC with an Intel Core i7 3.4GHz CPU and 16GB memory running Windows 7. All programs were implemented in Java, and the maximum main memory of the Java Virtual Machine is set to 4GB. The BIR-trees used are disk-resident. The page size is set to 4KB, and the fanout of the BIR-tree is set to 100. The buffer size is set at 4MB. For all algorithms, we measure the total query time (including the CPU time and IO time) and the IO cost. For each experiment, we randomly generate 1,000 queries and report the average performance.

Parameters

We vary different system parameters. These parameters together with their default values (highlighted in bold) are shown in Table 4.3. In the default setting, we select the object ranked $(10 \cdot k_0 + 1)st$ under $\bar{w}_0$ as the missing object.
3.5.2 Empirical Results

Scalability

To determine the scalability of our proposed algorithms, we randomly select different numbers, from 0.2M to 1.8M, of objects from the GN dataset to test the query performance under different dataset sizes. Fig. 3.8 shows that the BS and BP are superior to Baseline and TM in terms of both the total query time and IO cost. Moreover, BS and BP scale very well to large dataset sizes; the increase in their query time is sublinear when the dataset is enlarged. On the other hand, the IO cost of Baseline is several orders of magnitude higher than that of the other algorithms. Therefore, we omit it in the remaining experiments.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_0$</td>
<td>3, 10, 30, 100, 300</td>
</tr>
<tr>
<td># keywords</td>
<td>2, 4, 8, 16, 32</td>
</tr>
<tr>
<td>$\alpha_0$</td>
<td>$&lt;0.1, 0.9&gt;, &lt;0.3, 0.7&gt;, &lt;0.5, 0.5&gt;, &lt;0.7, 0.3&gt;, &lt;0.9, 0.1&gt;$</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.1, 0.3, 0.5, 0.7, 0.9</td>
</tr>
<tr>
<td># missing objects</td>
<td>1, 3, 10, 30</td>
</tr>
</tbody>
</table>

Figure 3.8: Varying Dataset Size
Varying $k_0$

In this set of experiments, we evaluate the effect of varying the parameter $k_0$ in the initial spatial keyword top-$k$ query. Note that the ranking of the missing object varies along with the change of $k_0$. For instance, when a top-3 spatial keyword query is posed initially, the corresponding why-not question is to ask why the object ranked 31 is missing; whereas a top-10 spatial keyword query corresponds to a why-not question for the object ranked 101. Fig. 4.4 plots the performance for the EURO dataset, showing that our algorithms are robust to changes in $k_0$. Also, we can see that TM has less IO than BS, but that its query time is much higher. As a result, TM is significantly worse than BS in terms of the total query time.

![Figure 3.9: Varying $k_0$](image)

Varying the number of query keywords

This experiment evaluates the effect of varying the number of query keywords. Theoretically, having more query keywords increases the CPU time needed to compute the similarity between the query keywords and the objects. However, as shown in Fig. 4.5, increasing the number of query keywords has little impact on the performance of BS and BP. This is because the textual similarity computations are simple for these two algorithms. On the other hand, in TM, as we need to process a progressive spatial keyword top-$k$ query many times, the computation of textual

---

3In the interest of space, we omit the results for the GN dataset as they are similar.
similarity becomes a dominating factor. This is why the query time of TM increases more noticeably with more query keywords.

![Graph showing query time and number of query keywords for TM, BS, and BP.](image)

Figure 3.10: Varying Number of Query Keywords

**Varying the initial weighting vector \( \vec{w}_0 \)**

Next we investigate the effect of varying the weighting vector \( \vec{w}_0 \) in the initial spatial keyword top-\( k \) query. As shown in Fig. 3.11, varying \( \vec{w}_0 \) has almost no impact on BS and BP. However, the query time of TM increases dramatically when the weight of \( ws \) is decreased. This indicates that BS and BP scale much better than TM w.r.t. the changing of initial weighting vector \( \vec{w}_0 \).

![Graph showing query time and number of query keywords for TM, BS, and BP with varying \( ws \) in initial weight vector.](image)

Figure 3.11: Varying Initial Weighting Vector \( \vec{w}_0 \)
Varying $\lambda$

Parameter $\lambda$ allows the user to indicate the preference of modifying $k_0$ or the weighting vector $\vec{w}_0$. When varying $\lambda$, we observe from Fig. 3.12 that BS and BP are almost unaffected. The reason is that in BS, $\lambda$ is only used in the last step when all promoted points are considered to compute the penalty. Changes to $\lambda$ do not affect the range query in BS or the number of promoted points. In BP, different $\lambda$ settings do not affect the pruning efficiency significantly. However, in TM, as the very basic refined query is to keep $\vec{w}_0$ and change $k_0$ to $R(o, q, \vec{w}_0)$, the penalty of this query is $\lambda$ according to Eqn. 3.1.3. Thus, a larger $\lambda$ causes a larger initial penalty in TM. This further influences the optimizations of TM mentioned in Section 3.5.1. Thus, TM is quite sensitive to changes in $\lambda$ and the query time increases dramatically with $\lambda$.

![Figure 3.12: Varying $\lambda$](image)

Varying the number of missing objects

Finally, we vary the number of the missing objects. In this experiment, the original query is a top-10 query, and the missing objects are chosen at random from the objects ranking between 11 and 301 under the original query. Fig. 4.9 shows that the query time of BS is quite stable as the number of missing objects increases. BP increases linearly in both query time and IO cost, but at a faster rate than BS. This is because after we identify all the working parts of the missing objects, the range query in BS still works well; but in BP, a node can only be pruned when it
does not affect all the working parts, and hence the pruning efficiency is degraded when more objects are missing. Yet, BP is still much better than TM in terms of the query time.

![Figure 3.13: Varying Number of Missing Objects](image)

3.6 Summary

In this chapter, we studied how to answer preference-adjusted why-not questions on spatial keyword top-k queries using query refinement. We reduced ranking updates to a geometrical problem, and based on this and the proposed BIR-tree, we developed basic and optimized bound-and-prune algorithms. Extensive empirical study on real datasets demonstrates that the proposed algorithms substantially outperform the existing ones in terms of both the query time and IO cost.

This chapter is a slightly amended version of the paper [8] that we published in ICDE 2015. The DOI of the paper is 10.1109/ICDE.2015.7113291.
Chapter 4

Keyword-Adapted Why-Not
Spatial Keyword Top-$k$ Query

This chapter studies the problem of keyword-adapted why-not spatial keyword top-$k$ queries, which aim to provide users with more precise query keywords that better capture their query intents. The chapter is organized as follows. Section 4.1 describes preliminaries and defines the keyword-adapted why-not spatial keyword top-$k$ query. Section 4.2 presents the basic query algorithm along with a set of optimizations. Section 4.3 proposes an index-based solution. We extend the algorithm to support queries with multiple missing objects in Section 4.4. Experimental results are reported in Section 4.5, followed by a conclusion in Section 4.6.

4.1 Preliminaries and Problem Definition

4.1.1 Preliminaries

We first review the spatial keyword top-$k$ query. Let $\mathcal{D}$ denote a database of spatial objects. Each object $o \in \mathcal{D}$ is a pair $(o.loc, o.doc)$, where $o.loc$ is a point location and $o.doc$ is a set of generic keywords that describe the object. A spatial keyword top-$k$ query $q$ retrieves $k$ top ranked objects from $\mathcal{D}$ according to a ranking function that takes into consideration both spatial distance and textual similarity. For broad
applicability, we consider a widely used ranking function [14]:

\[ ST(o, q) = \alpha \cdot (1 - SDist(o, q)) + (1 - \alpha) \cdot TSim(o, q), \]  

(4.1.1)

where \( SDist(o, q) \) denotes spatial distance normalized by the maximum possible distance between two points in \( D \), \( TSim(o, q) \) denotes textual similarity, and \( \alpha \in (0, 1) \) represents the user’s relative preference between spatial distance and textual similarity.

A spatial keyword top-\( k \) query \( q \) is a 4-tuple \((loc, doc, k, \alpha)\), where \( q.loc \) is a query point location, \( q.doc \) is a set of query keywords, \( q.k \) is the number of objects to retrieve, and \( q.\alpha \) denotes the user preference. The distance \( SDist(o, q) \) is calculated as the (normalized) Euclidean distance between \( o.loc \) and \( q.loc \). The textual similarity \( TSim(o, q) \) can be computed using an information retrieval model. Without loss of generality, we adopt the widely-used Jaccard similarity model [30]:

\[ TSim(o, q) = \frac{|o.doc \cap q.doc|}{|o.doc \cup q.doc|} \]  

(4.1.2)

In the ranking function, the higher the score computed by Eqn. (4.1.1), the higher the ranking of the corresponding object. We define the rank of an object \( o \) as follows:

\[ R(o, q) = |\{o' \in D|ST(o', q) > ST(o, q)\}| + 1 \]  

(4.1.3)

A spatial keyword top-\( k \) query \( q \) returns a set \( R \) of \( k \) objects from \( D \), where \( \forall o \in R \ (\forall o' \in D - R \ (ST(o, q) \geq ST(o', q))) \), or in terms of object rank, \( \forall o \in R \ (\forall o' \in D - R \ (R(o, q) \leq R(o', q))) \).

4.1.2 Keyword-Adapted Why-Not Spatial Keyword Query

It is often difficult for users to find the keywords that best describe their query intention. Thus, identifying the proper query keyword set is arguably the key

\footnote{The algorithm developed in Section 4.2 can support other similarity models by using the particular spatial keyword top-\( k \) query algorithms associated with these models; the KeR-tree based algorithm proposed in Section 4.3 can also be extended to support other models, e.g., the Dice coefficient and the Cosine similarity.}

45
challenge in issuing a spatial keyword top-$k$ query. After a user issues a query $q = (\text{loc}, \text{doc}_0, k_0, \alpha)$ and receives the result, the user may observe that one or more objects that were expected to be in the result are missing. The user may then pose a why-not question with a set of missing objects $M = \{m_1, m_2, ..., m_j\}$, asking the system for a refined query $q' = (\text{loc}, \text{doc}', k', \alpha)$ the result of which contains the missing objects. Since it is possible that no modified set of keywords can revive the missing objects, we also consider the enlargement of $k$. Different modifications of these two parameters may yield many qualified queries that retrieve the missing objects, and we prefer the refined query that minimally modifies the original query. Specifically, we adopt a penalty model [17, 19] that associates a penalty with a refined query. It is defined as the weighted sum of the modifications of the two parameters, i.e., $\Delta k$ and $\Delta \text{doc}$. The penalty (cost) of a query $q'$ that refines an original query $q$ is defined as follows:

$$\text{Penalty}(q, q') = \lambda \cdot \frac{\Delta k}{R(M, q) - k_0} + (1 - \lambda) \cdot \frac{\Delta \text{doc}}{|\text{doc}_0 \cup M.\text{doc}|}$$

(4.1.4)

Here, $\lambda$ is a user preference on the modification of $q.k$ versus $q.doc$ and $R(M, q) = \max_{m_i \in M} R(m_i, q)$. Next, $\Delta k = \max(0, k' - k_0)$ since for a refined query $q'$, if $R(M, q') > k_0$, $k'$ must be no smaller than $R(M, q')$ to revive the missing objects; otherwise, $k'$ can remain at $k_0$. As in other studies [19], we normalize $\Delta k$ by $R(M, q) - k_0$, as a basic refined query is to keep the original query keywords and enlarge $k_0$ to $R(M, q)$; for other refined queries that modify the query keywords to achieve a lower penalty than that of this basic one, $\Delta k$ must not exceed $R(M, q) - k_0$. Using the principle of edit distance, the modification of query keywords $\Delta \text{doc}$ is quantified as the minimum count, denoted as $ED(\text{doc}_0, \text{doc}')$, of edit operations needed to transform $\text{doc}_0$ to $\text{doc}'$. For simplicity, we consider two edit operations: insertion and deletion. Similarly, we normalize $ED(\text{doc}_0, \text{doc}')$ by the maximum possible number of edit operations needed to modify $\text{doc}_0$ into a $\text{doc}'$ that yields a query that retrieves all objects in $M$. This quantity is estimated as $|\text{doc}_0 \cup M.\text{doc}|$, where $M.\text{doc} = \bigcup_{i=1}^{j} m_i.\text{doc}$. In other words, we just consider the keywords in $M.\text{doc}$, as adding a keyword not in $M.\text{doc}$ would make the set of query keywords less relevant.
to the user’s query intention, i.e., less relevant to the missing objects.

We now define the keyword-adapted why-not spatial keyword top-

\[ k \] queries as follows:

**Definition 4.1.1. Keyword-Adapted Why-Not Spatial Keyword Top-k Query.**

Given a set \( \mathcal{D} \) of spatial objects, a missing object set \( M \subset \mathcal{D} \), an original spatial keyword query \( q = (\text{loc}, \text{doc}_0, k_0, \alpha) \), the keyword-adapted why-not spatial keyword top-k query returns the refined query \( q' = (\text{loc}, \text{doc}', k', \alpha) \), with the lowest penalty according to Eqn. (4.1.4) and the result of which contains all objects in \( M \).

<table>
<thead>
<tr>
<th>Table 4.1: An Example of Refined Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refined Query</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>( q_1 = (3, {t_1, t_2}) )</td>
</tr>
<tr>
<td>( q_2 = (1, {t_2, t_3}) )</td>
</tr>
<tr>
<td>( q_3 = (2, {t_1, t_3}) )</td>
</tr>
<tr>
<td>( q_4 = (2, {t_1, t_2, t_3}) )</td>
</tr>
</tbody>
</table>

**Example 5.** Fig. 4.1 illustrates the keyword-adapted why-not query. Figure 4.1(a) shows the locations and keywords of four objects and the initial query \( q \). Figure 4.1(b) lists the values of \( 1 - SDist(o, q) \), \( TSim(o, q) \) as well as the ranking score \( ST(o, q) \) of each object with respect to (w.r.t.) the initial query that has \( \text{doc}_0 = \{t_1, t_2\} \), \( k_0 = 1 \), and \( \alpha = 0.5 \). According to the \( ST(o, q) \) values, we can see that object \( m \) has rank 3 under the initial query, so it is not in the result. Table 4.1
lists several refined queries that retrieve \( m \) together with their penalties, where \( \lambda = 0.5 \). We omit \( \text{loc} \) and \( \alpha \) in the refined queries as they stay unchanged. In this setting, \( R(m, q) - k_0 = 2 \) and \( |\text{doc}_0 \cup m.\text{doc}| = 3 \). Query \( q_1 \) keeps the initial query keywords and enlarges \( k_0 \) to \( R(m, q) \). So the total penalty is \( 0.5 \cdot \frac{3-1}{2} + 0.5 \cdot \frac{0}{3} = 0.5 \). Similarly, \( q_2 \) keeps \( k_0 \) and changes the query keywords to \( \{t_2, t_3\} \), which causes a smaller penalty. Query \( q_2 \) is the best refined query.

### 4.2 Basic Why-not Answering Algorithm

Following the problem analysis, we present a basic algorithm and then several optimizations.

#### 4.2.1 Problem Analysis

Recall that we consider refining the query keywords and \( k \) to achieve the inclusion of missing objects. Only refined pairs \( (\text{doc}', k') \) that satisfy Lemma 4.2.1 are candidates for the best refined query.

**Lemma 4.2.1.** Given a set \( M \) of missing objects, a pair of refined query keywords and a result cardinality \( (\text{doc}', k') \) can result in the best refined query if and only if (i) \( k' = R(M, q') \), when \( R(M, q') \geq k_0 \); or (ii) \( R(M, q') \leq k' \leq k_0 \), when \( R(M, q') < k_0 \), where \( R(M, q') = \max_{m_i \in M} R(m_i, q') \).

The proof is straightforward and hence omitted. Lemma 4.2.1 implies that given a refined keyword set \( \text{doc}' \), we may set \( k' = R(M, q') \) to get the minimum penalty. This observation inspires a basic solution to the keyword-adapted why-not query. In the following, we first consider a single missing object \( m \). In Section 4.4, we show how the algorithms can be adapted to handle multiple missing objects.

#### 4.2.2 Basic Algorithm

The basic algorithm works as follows. First, we determine the rank \( R(m, q) \) of the missing object \( m \) w.r.t. the initial spatial keyword query by processing the query
until object \( m \) appears. Then we enumerate all possible query keyword sets and invoke an existing spatial keyword top-\( k \) query algorithm (e.g., [14]) for each such set to determine the ranking of object \( m \) for each keyword set. Finally, we return the keyword set with the minimum penalty according to Eqn. (4.1.4).

To support spatial-keyword top-\( k \) query processing with the Jaccard similarity model, similar to related work [14, 32], we employ a hybrid index that estimates bounds on spatial distance and textual similarity at the same time. This index, called the SetR-tree, is a variant of the IR-tree [26]. A leaf node of the SetR-tree stores a number of entries of the form \((o, mbr, pks)\), where \( o \) represents an object, \( mbr \) is the minimum bounding rectangle (MBR) of the object, and \( pks \) is a pointer to the keyword set of \( o \). A non-leaf node stores a number of entries of the form \((pc, mbr, pku, pki)\), where \( pc \) is a child pointer, \( mbr \) is the MBR of the child node, \( pku \) is a pointer to the union of the keyword sets of all the objects indexed in the subtree rooted at this node, and \( pki \) is a pointer to the intersection of the keyword sets. These two sets are stored sequentially on disk to reduce the number of disk seeks. Fig. 4.2 shows an example SetR-tree, where the keyword sets associated with each leaf and non-leaf node are shown. For instance, consider the non-leaf node \( R_3 \); its union (resp., intersection) set is the union (resp., intersection) of keyword sets associated with leaf nodes \( R_1 \) and \( R_2 \).

![Figure 4.2: An Example SetR-tree](image)

**Theorem 4.2.2.** Consider a query \( q = (loc, doc, k, \alpha) \) and a SetR-tree node \( N \) that indexes a set \( S \) of objects. Let \( MDist(q.loc, N.MBR) \) denote the minimum distance...
between q.loc and N’s MBR and let \( N_\cup \) and \( N_\cap \) denote the union and intersection keyword sets of N, respectively. It holds that:

\[
\forall o \in S \ (ST(o, q) \leq \alpha \cdot (1 - MDist(q.loc, N.MBR)) + (1 - \alpha) \cdot \frac{|N_\cup \cap q.doc|}{|N_\cap \cup q.doc|})
\]

(4.2.5)

\textbf{Proof.} As \( o \) is enclosed by \( N.MBR \), the distance between \( q.loc \) and \( o.loc \) must be no less than \( MDist(q.loc, N.MBR) \), which is to say:

\[
1 - SDist(o.loc, q.loc) \leq 1 - MDist(q.loc, N.MBR)
\]

Next, \( o.doc \in N_\cup \) and \( N_\cap \in o.doc \), which ensures that \( o.doc \cap q.doc \in N_\cup \cap q.doc \) and \( N_\cap \cup q.doc \in o.doc \cup q.doc \). So we have:

\[
TSim(o.doc, q.doc) = \frac{|o.doc \cap q.doc|}{|o.doc \cup q.doc|} \leq \frac{|N_\cup \cap q.doc|}{|N_\cap \cup q.doc|}
\]

Recall that \( ST(o, q) = \alpha \cdot (1 - SDist(o.loc, q.loc)) + (1 - \alpha) \cdot TSim(o.doc, q.doc) \), where neither of the two terms exceeds those of Eqn. (4.2.5). Thus, Theorem 4.2.2 holds.

Theorem 4.2.2 enables us to estimate the bounds for spatial distance and textual similarity using the SetR-tree, which further provides the bound of the ranking score of each object indexed by a SetR-tree. This facilitates the search by a SetR-tree to find the top-k most relevant objects. The spatial keyword query processing algorithm using the SetR-tree is similar to that of the IR-tree [26]. Interested readers may refer to [26] for details.

One other issue we need to address is the possible query keyword sets. As the total number of distinct keywords in \( D \) may be huge and adding a keyword that is not in \( m.doc \) would make the set of query keywords less relevant to the user’s query intention, we consider only the keywords in \( m.doc \).

### 4.2.3 Optimizations

The above algorithm essentially executes a spatial keyword query for each candidate keyword set \( doc' \) to determine the ranking of the missing object for the set. The
doc' sets are obtained by adding keywords in m.doc to doc0 and by deleting keywords originally in doc0. This yields \(2^{[doc0 \cup m.doc]}\) candidate keyword sets. For each such keyword set, a spatial keyword query is processed until the missing object is retrieved. We now present techniques that accelerate this process.

**Early Stop**

In the basic algorithm, for each doc', we need to invoke a spatial keyword query to determine the ranking of the missing object, which means that the query can stop only when the missing object \(m\) appears in the result. However, for a keyword set doc' that is not very relevant to \(m\), the number of objects that rank higher than \(m\) may be very large. Fortunately, such extensive computations can be avoided. We have seen that a basic refined query is to just enlarge \(k\) in the original query to \(R(m, q)\) without modifying the query keyword set. This refined query has penalty \(\lambda\) according to Eqn. (4.1.4). Our approach is to maintain a currently best refined query together with its penalty \(p_c\). The penalty of a refined query is the sum of two terms: the penalty for modifying \(k_0\) and the penalty for modifying \(doc_0\). Given a keyword set doc', we can compute \(\Delta doc\) accordingly. The query \(q'\) generated from doc' can be the best refined query if and only if:

\[
\lambda \cdot \frac{\max(0, R(m, q') - k_0)}{R(m, q) - k_0} + (1 - \lambda) \cdot \frac{\Delta doc}{|doc0 \cup m.doc|} \leq p_c,
\]

from which we can deduce the ranking lower bound \(R_L(m, q')\) for the missing object \(m\) w.r.t. a refined keyword set doc':

\[
R_L(m, q') = [k_0 + \frac{p_c - (1 - \lambda) \cdot \frac{\Delta doc}{|doc0 \cup m.doc|} \cdot (R(m, q) - k_0)}{\lambda}]
\]

Eqn. (4.2.6) implies that we can stop processing the spatial keyword query corresponding to a keyword set doc' as soon as we retrieve more than \(R_L(m, q')\) objects. When a new query \(q'\) is found that has a smaller penalty than the currently smallest penalty, \(p_c\) is updated and query \(q'\) is recorded as the currently best refined query.

**Example 6.** Consider a top-5 query, where the ranking \(R(m, q)\) of the missing object under the initial query is 10 and \(\lambda = 0.5\). Assume that the currently best
refined query \( q_c \) has penalty \( p_c = 0.5 \). Now we are going to process a candidate keyword set \( doc' \) w.r.t. which the ranking of the missing object is 100. Suppose 
\[
\Delta_{doc} = 0.4.
\]
Then according to Eqn. (4.2.6), we can calculate 
\[
R_L(m, q') = [5 + \frac{0.5 - (1 - 0.5) \cdot 0.4}{0.5} \cdot (10 - 5)] = 8,
\]
which means we can stop the processing of the query generated by \( doc' \) as soon as we retrieve 8 objects.

**Enumeration Order**

The order in which we consider the candidate keyword sets plays an important role in the basic algorithm. First, \( p_c \) plays a key role in the pruning, which further determines \( R_L(m, q') \) for an incoming \( doc' \). Finding a small \( p_c \) early can speed up the pruning and improve the efficiency of the algorithm. Second, the dominating objects of the missing object in the previous queries have high chances to keep dominating \( m \) if the next keyword set is similar to the previous ones. Thus, we consider the keyword sets in a particular order. We should first consider the keyword sets who are more likely to be the final best refined query. We estimate this “more likely” based on two conjectures: keyword sets with a smaller edit distance to the original keyword set are “more likely” since they incur lower penalty; and adding a keyword that is more particular to a missing object will make the query more related to \( m \).

We define the particularity of a keyword \( t \) to an object \( o \) using the IDF (inverse document frequency) weight as follows:

\[
Parti(o, t) = \begin{cases} 
-\log \frac{|D| - n_t + 0.5}{n_t + 0.5} & t \notin o.doc \\
\log \frac{|D| - n_t + 0.5}{n_t + 0.5} & t \in o.doc 
\end{cases},
\]

where \( n_t \) is the number of objects that contain the keyword \( t \). Thus, we consider the candidate keyword sets in increasing order of their edit distance to \( doc_0 \). Further, sets with the same edit distance are considered in ascending order of the sum of the total particularity of the inserted (+) and deleted (-) keywords. This ordering also provides an early termination condition on the enumeration process: when the current \( p_c \) is less than the penalty of the next keyword set in the modification of keywords, we can guarantee no remaining keyword sets can be the best refined set,
and we can return the currently best refined set as the final solution.

**Keyword Set Filtering**

Since similar keyword sets have similar textual similarities to an object, the rankings of objects according to the scoring function (Eqn. (4.1.1)) may be similar for similar query keyword sets. For instance, if we have processed a spatial keyword query for a keyword set $doc_0$ and have obtained the set $C$ of objects that rank higher than missing object $m$ (i.e., dominators of $m$) then for the keyword set $doc'$ that is obtained by slightly modifying $doc_0$, e.g., removing a keyword from $doc_0$, the textual similarity and ranking for each object in $C$ will change very little. Thus, the objects in $C$ will stand a good chance to keep ranking higher than $m$. Therefore, we propose to cache the dominators of the missing object retrieved from the previously processed query. Before generating and processing a spatial keyword top-$k$ query for a new keyword set $doc'$, we test how many cached dominators still rank higher than the missing object. If this number exceeds the lower bound of the ranking we derived for $doc'$ using Eqn. (4.2.6), $doc'$ can be pruned safely without further processing.

**Parallel Processing**

Another observation is that the processing of spatial keyword queries generated from different candidate keyword sets is independent of each other, with the exception that the parameters such as $p_c$ and $R_L$ need to be synchronized to stop early. As such, we also optimize the algorithm by concurrently processing the queries using multiple physical threads available in the underlying machine.

Algorithm 3 summarizes the basic algorithm along with the optimizations.

### 4.3 Bound-and-Prune Algorithm

While the optimizations make the basic algorithm more efficient, it still needs to invoke a spatial keyword query for each candidate keyword set one by one, where
Algorithm 3 Basic Algorithm for Answering Why-not Questions

INPUT: SetR-tree $\mathcal{T}$, original query $q = (\text{loc}, \text{doc}_0, k_0, \alpha)$, missing object $m$

OUTPUT: Best refined query $q' = (\text{loc}, \text{doc}', k', \alpha)$

1: determine $R(m, q)$
2: $\text{doc}' \leftarrow \text{doc}_0, k' \leftarrow R(m, q), p_c \leftarrow \lambda$ //initialize the best refined query and the penalty threshold using the basic refined query
3: $S \leftarrow \emptyset$ // a candidate query keyword set
4: repeat
5: $S \leftarrow \text{NextKeywordSet}()$ // find the next query keyword set according to the enumeration order
6: if $(1 - \lambda) \cdot \frac{\Delta \text{doc}}{|\text{doc}_0 \Delta m \cdot \text{doc}|} \geq p_c$
7: break
8: compute $R_L$ for $S$ according to Eqn. (4.2.6)
9: $t \leftarrow 0$
10: for each object $o$ in $C$
11: if $ST(o, q_S) > ST(m, q_S)$ then $t++$ // $q_S$ is the spatial keyword query generated by the keyword set $S$
12: if $t \geq R_L$
13: continue
14: determine $R(m, S)$ by processing a spatial keyword query // in query processing, if the number of retrieved objects exceeds $R_L$, the processing terminates and the algorithm continues to the next keyword set
15: compute the penalty $p$ for $S$ according to Eqn. (4.1.4)
16: if $p < p_c$ then
17: $p_c \leftarrow p, \text{doc}' \leftarrow S, k' \leftarrow R(m, S)$
18: until $S = \emptyset$
19: return $q' = (\text{loc}, \text{doc}', k', \alpha)$

each time multiple SetR-tree nodes need to be loaded into memory and accessed. As I/O operations are much more costly than in-memory operations, loading the index
nodes and data objects for each keyword set is wasteful. Also, the intersection/union keyword sets in an upper node of the SetR-tree may be empty/large if the underlying objects are from different categories, which would further slow down the processing of a spatial keyword query. To address these problems, we present an index-based bound-and-prune algorithm that finds the best sample out of a number of keyword sets and prunes the others in just one index access. As the index nodes and data objects are loaded just once for a set of keyword sets, substantial I/O can be saved.

4.3.1 Preliminaries: KcR-Tree

We adopt the KcR-tree (Keyword count R-tree) [28] as the index structure. Essentially, a KcR-tree is an R-tree that integrates the textual information of the indexed objects into each tree node.

A leaf node of the KcR-tree contains entries of the form \((o, mbr, pks)\), where \(o\) represents an object, \(mbr\) is the minimum bounding rectangle of the object, and \(pks\) is a pointer to the keyword set of the object. A non-leaf node contains entries of the form \((pc, mbr, pcm)\), where \(pc\) is a pointer to a child node, \(mbr\) is the minimum bounding rectangle of the child node, and \(pcm\) is a pointer to a keyword-count map \(kcm\) of the child node. More specifically, \(kcm\) is a key-value map where each key is a keyword corresponding to the objects in the child node and each value is the number of objects in the child node that contain this keyword. In addition, each node of the KcR-tree stores a \(cnt\) value, which is the total number of the objects indexed in the subtree rooted at this node. Fig. 4.3 illustrates a KcR-tree. For example, in \(R_1\), the value \(cnt = 3\) means that the subtree of this node indexes 3 objects, \(i.e., o_1, o_2, o_3\). The \(kcm\) of this node has keys for \(o_1.doc \cup o_2.doc \cup o_3.doc = \{\text{Chinese, restaurant}\}\). Two of the three objects have the keyword \(\text{Chinese}\) and all of them have the keyword \(\text{restaurant}\).
4.3.2 Properties of KcR-tree

Given a KcR-tree node \( N \), for a query keyword set \( S \), we can estimate the upper and lower bounds on the number of objects in \( N \) that rank higher than the missing object \( m \). Let \( MaxDom(N, S, m) \) and \( MinDom(N, S, m) \) denote the upper and lower bounds, respectively. We proceed to show how such bounds are estimated.

**Theorem 4.3.1.** Given a KcR-tree node \( N \), an initial query \( q \), a missing object \( m \), and a refined query keyword set \( S \), if an object \( o \) indexed in \( N \) ranks higher than \( m \) under the new query, the following inequality must hold:

\[
TSim(o, S) > \frac{\alpha}{1 - \alpha} \cdot (MinDist(N, q) - SDist(m, q)) + TSim(m, S),
\]

where \( MinDist(N, q) \) is the minimum distance between the query location \( q.loc \) and \( N \)'s minimum bounding rectangle.

**Proof.** Under the new query, if an object \( o \) in node \( N \) ranks higher than \( m \), it holds that:

\[
\alpha \cdot (1 - SDist(o, q)) + (1 - \alpha) \cdot TSim(o, S) > \alpha \cdot (1 - SDist(m, q)) + (1 - \alpha) \cdot TSim(m, S),
\]

which can be rewritten as:

\[
TSim(o, S) > \frac{\alpha}{1 - \alpha} \cdot (SDist(o, q) - SDist(m, q)) + TSim(m, S)
\]
Since object $o$ is enclosed by node $N$’s minimum bounding rectangle, the following is true:

$$SDist(o, q) > MinDist(N, q)$$

The theorem follows by combining the two inequalities.

Theorem 4.3.1 provides a lower bound of the textual similarity for an object $o$ in $N$ that ranks higher than the missing object w.r.t. query keyword set $S$. We denote the lower bound as $TSim(N, o, S)_L$. Next, we introduce another important metric, the pseudo textual similarity between a KcR-tree node $N$ and a keyword set $S$, which will be used in the estimation of the ranking bounds.

**Definition 4.3.2.** The pseudo textual similarity between a KcR-tree node $N$ and a keyword set $S$ is defined as follows:

$$TSim^\sim(N, S) = \frac{\sum_{t \in S \cap N.doc} N.count(t)}{|S| \cdot N.cnt + \sum_{t \in N.doc - S} N.count(t)}$$

where $N.doc$ denotes the keyword set in $N$’s keyword-count map and $N.count(t)$ denotes the counting of term $t$ in node $N$.

The following theorem establishes a relationship between $TSim(N, o, S)_L$ and the pseudo textual similarity.

**Theorem 4.3.3.** Given a KcR-tree node $N$, an initial query $q$, a missing object $m$, and a refined keyword set $S$, all the objects indexed in $N$ are dominators of $m$ if and only if:

$$TSim^\sim(N, S) \geq TSim(N, o, S)_L$$

**Proof.** Based on Theorem 4.3.1, to ensure each object indexed in $N$ is a dominator of $m$, it holds that $\forall o \in N$ ($TSim(o, S) > TSim(N, o, S)_L$). According to the definition of textual similarity, $TSim(o, S) = \frac{|o.doc \cap S|}{|o.doc \cup S|} \geq TSim(N, o, S)_L$. Therefore, for each object $o \in N$, the inequality $|o.doc \cap S| \geq |o.doc \cup S| \cdot TSim(N, o, S)_L$ holds, from which we can further deduce that:

$$\sum_{o \in N} |o.doc \cap S| \geq \sum_{o \in N} |o.doc| \cdot TSim(N, o, S)_L$$
Since \( N.doc = \bigcup_{o \in N} o.doc \), we have \( \sum_{o \in N} |o.doc \cap S| = \sum_{t \in S \cap N.doc} N.count(t) \) and 
\( \sum_{o \in N} |o.doc \cup S| = |S| \cdot N.cnt + \sum_{t \in N.doc - S} N.count(t) \). Thus, \( \sum_{t \in S \cap N.doc} N.count(t) \geq \left( |S| \cdot N.cnt + \sum_{t \in N.doc - S} N.count(t) \right) \cdot TSim(N, o, S)_L \), which can be further rewritten as \( TSim^\sim(N, S) = \frac{\sum_{t \in S \cap N.doc} N.count(t)}{|S| \cdot N.cnt + \sum_{t \in N.doc - S} N.count(t)} \geq TSim(N, o, S)_L \). The theorem follows.

Theorem 4.3.3 implies that, given a KcR-tree node \( N \), if the inequality \( TSim^\sim(N, S) \geq TSim(N, o, S)_L \) does not hold, at least one object in \( N \) ranks lower than the missing object \( m \). This suggests a possibility of estimating \( MaxDom(N, S, m) \) and allows us to develop Algorithm 4 to derive it. We first assume that all objects in \( N \) dominate the missing object \( m \), i.e., \( MaxDom(N, S, m) = N.cnt \) (Line 4). We then iteratively virtually prune objects from \( N \) until Theorem 3 holds for the consequent virtual node and return the number of remaining objects as the upper bound on \( m \)'s dominators in \( N \) (Lines 5–14). As we do not know the keyword sets of each object in \( N \), to find the upper bound \( MaxDom(N, S, m) \), we assume all the virtually pruned objects are irrelevant to the set of query keywords. We associate the irrelevant (resp., relevant) keywords with the pruned (resp., remaining) objects as many as possible. To do so, we divide the keywords in \( N.doc \) into two categories, i.e., \( N.doc - S \) and \( N.doc \cap S \). The keywords in \( N.doc \cap S \), which are relevant to the query keywords, are kept for the remaining objects, except we have to associate them with the pruned ones, i.e., \( N.count(t) > ans \) (Line 11). The keywords in \( N.doc - S \), which are irrelevant to the query keywords, are pruned as long as we can associate them with the pruned object, i.e., the keyword set \( T_{N-S} \leftarrow \{ t \mid t \in N.doc - S \land N.count(t) \geq (N.cnt - ans) \} \) (Line 12). We omit the details of the estimation of \( MinDom(N, S, m) \) as it is done similarly.

**Example 7.** Consider a KcR-tree node \( N \), whose keyword-count map is \( \{(t_1, 8), (t_2, 3), (t_3, 7), (t_4, 2), (t_5, 1)\} \) and \( N.cnt = 8 \). Assume the query keyword set \( S = \{t_3, t_4\} \) and that we have computed \( TSim(N, S, o)_L = 0.395 \). In this setting, \( S \cap N = \{t_3, t_4\} \) and \( N - S = \{t_1, t_2, t_5\} \). We start by assuming all the objects indexed in \( N \) rank higher than the missing object \( m \) w.r.t. \( S \), i.e., \( MaxDom(N, S, m) = 8 \). We
Algorithm 4 MaxDom(N,S,m)

INPUT: A KcR-tree Node N, a keyword set S, the missing object m

OUTPUT: MaxDom(N, S, m)

1: calculate $TSim(N, o, S)_L$
2: $C_{S \cap N} \leftarrow \sum_{t \in S \cap N.doc} N.count(t)$
3: $C_{N - S} \leftarrow \sum_{t \in N.doc - S} N.count(t)$
4: $ans \leftarrow N.cnt$
5: while $ans > 0$ do
6: $TSim^~(N, S) \leftarrow \frac{C_{S \cap N}}{|S| - ans + C_{N - S}}$
7: if $TSim^~(N, S) \geq TSim(N, o, S)_L$ then
8: return $ans$
9: else
10: $ans \leftarrow ans - 1$
11: $T_{S \cap N} \leftarrow \{t | t \in N.doc \cap S \cap N.count(t) > ans\}$
12: $T_{N - S} \leftarrow \{t | t \in N.doc - S \cap N.count(t) \geq (N.cnt - ans)\}$
13: $C_{S \cap N} \leftarrow C_{S \cap N} - |T_{S \cap N}|$
14: $C_{N - S} \leftarrow C_{N - S} - |T_{N - S}|$

try to verify our assumption using Theorem 4.3.3. We find that $TSim^~(N, S) = \frac{9}{28} < TSim(N, S, o)_L$. Our assumption fails. Then we modify our assumption on MaxDom(N, S, m) and virtually prune an object $o$ from the node by associating as many irrelevant keywords as possible with $o$, which leads to a virtual KcR-tree node $N'$ that indexes 7 objects and has keyword-count map $\{(t_1, 7), (t_2, 2), (t_3, 7), (t_4, 2), (t_5, 0)\}$. Again, we test the assumption and find $TSim^~(N, S) = \frac{9}{23}$, which is still less than 0.395. Hence, we prune one more object from $N'$. However, as $N'.count(t_5) = 0$, we cannot associate $t_5$ with the pruned objects; on the other hand, as $N'.count(t_3) = N'.cnt$, we have to associate it with the pruned object. After this iteration, the pseudo textual similarity between the consequent node and $S$ is 0.4, which exceeds $TSim(N, S, o)_L$. Thus, the function terminates and returns 6 as MaxDom(N, S, m).
4.3.3 Optimized Bound-and-Prune Algorithm

As shown above, given a KcR-tree node $N$, a refined keyword set $S$ (a refined query), and the missing object $m$, we can estimate the upper and lower bounds on the number of objects indexed in $N$ that dominate $m$ w.r.t. $S$, i.e., $MaxDom(N, S, m)$ and $MinDom(N, S, m)$. Based on this, we propose an optimized bound-and-prune algorithm.

The basic idea is as follows. Given a set $CK$ of candidate keyword sets, we traverse the KcR-tree $T$ starting from the root. For each candidate keyword set $S$ in $CK$, we maintain the upper and lower bounds of the missing object $m$’s ranking under $S$, i.e., $\hat{R}(S, m)$ and $\hat{\hat{R}}(S, m)$, which are initially estimated as $MaxDom(T.root, S, m) + 1$ and $MinDom(T.root, S, m) + 1$. By knowing $m$’s ranking bounds w.r.t. a keyword set $S$, we can compute the penalty upper bound and lower bound on the candidate keyword set $S$, i.e., $\hat{p}n(S)$ and $\hat{\hat{p}}n(S)$. When we traverse the KcR-tree downwards and access lower level nodes, the penalty bounds $\hat{p}n(S)$ and $\hat{\hat{p}}n(S)$ of a candidate keyword will be tightened gradually along with updates of $\hat{R}(S, m)$ and $\hat{\hat{R}}(S, m)$. We can safely prune a keyword set $S$ if $\hat{p}n(S)$ exceeds the penalty of the known best refined query. And we can replace the known best refined query with the query generated by $S$ if $\hat{p}n(S)$ is less than its penalty.

The optimized bound-and-prune algorithm determines the best refined query keyword set among a set of candidates in just one traversal of the KcR-tree. It estimates the missing object’s ranking as well as the penalty w.r.t. each candidate keyword set $S$ before actually unfolding a KcR-tree node. The algorithm is detailed in Algorithm 5. It takes as input a currently known best refined query $q’$ and a set of candidate keyword sets $CK$, and it returns the best refined query among the input $q’$ and the queries generated by each query keyword set in $CK$. The currently best refined query is initialized to the basic refined query that just sets the number of retrieved objects to $R(m, q)$ and keeps the initial query keywords. Let $\hat{D}(N, S)$ and $\hat{\hat{D}}(N, S)$ denote the upper and lower bounds of the number of objects in a KcR-tree node $N$ that rank higher than the missing object $m$, which can be computed by
the two functions \( \text{MaxDom}(N, S, m) \) and \( \text{MinDom}(N, S, m) \), respectively. First, we estimate the upper bound and lower bound of the ranking of the missing object \( m \) w.r.t. each candidate keyword set \( S \) as \( \hat{D}(T.\text{root}, S) + 1 \) and \( \hat{D}(T.\text{root}, S) + 1 \) (Lines 2–6). Then we insert the root \( T \) into the queue \( Q \) (Line 7) and start to traverse the KcR-tree. In each iteration, we dequeue a KcR-tree node \( N \) from \( Q \) (Line 9). By unfolding \( N \), we obtain more detailed spatial and textual information. We next access each child \( c \) of \( N \) and compute \( \hat{D}(c, S) \) and \( \hat{D}(c, S) \) by calling the \( \text{MaxDom}(c, S, m) \) and \( \text{MinDom}(c, S, m) \) functions (Lines 14–15). By doing so, we get tighter upper and lower bounds of objects in \( N \) that rank higher than the missing object, i.e., \( \hat{D}'(N, S) = \sum_{c \in N} \hat{D}(c, S) \) and \( \hat{D}'(N, S) = \sum_{c \in N} \hat{D}(c, S) \) (Lines 16–17). Obviously, the difference between \( \hat{D}(N, S) \) and \( \hat{D}'(N, S) \) is the number of objects that are considered wrongly to rank higher than \( m \) when we just access \( N \). Thus, we can re-estimate the upper bound of \( m \)’s ranking w.r.t \( S \) more accurately by subtracting this difference (Line 18). The tightened lower bound of the ranking of \( m \) under \( S \) can be computed similarly (Line 19). With the tighter bounds on \( m \)’s ranking w.r.t \( S \), we can compute its penalty upper bound and lower bound according to the penalty function Eqn. (4.1.4) (Line 20). We update the currently known best refined query with the query generated by \( S \) if the penalty upper bound \( \hat{pn}(S) \) of \( S \) is smaller than \( p_c \) (Lines 21–23). Otherwise, we prune a candidate \( S \) if \( \hat{pn}(S) > p_c \) (Lines 25–26). Also, we prune the nodes that cannot further tighten the bounds (Lines 29–30). We return the best refined query after \( Q \) or \( CK \) becomes empty (Lines 27–28, Line 33).

4.3.4 Strategic Use of Algorithm 5

Algorithm 5 finds the best refined query among a set of candidate keyword sets. One straightforward way of using Algorithm 5 to answer a why-not query is to first generate all the candidate keyword sets and input them to the algorithm. However, the performance of Algorithm 5 is dominated by the execution time of the \text{MaxDom} and \text{MinDom} functions, which is proportional to the number of candidate keyword
Algorithm 5 KcR-tree Based Algorithm for Answering Why-not Questions

INPUT: KcR-tree \( T \), current best refined query \( q' = (loc, doc', k', \alpha) \), candidate keyword sets \( CK \), missing object \( m \), current best penalty \( p_c \)

OUTPUT: Best refined query \( q' = (loc, doc', k', \alpha) \) among keyword sets in \( CK \) and its penalty \( p_c \)

1: \( Q \leftarrow \) empty queue
2: \textbf{for} each keyword set \( S \) in \( CK \) \textbf{do}
3: \( \hat{D}(T.\text{root}, S) \leftarrow \text{MaxDom}(T.\text{root}, S, m) \)
4: \( \check{D}(T.\text{root}, S) \leftarrow \text{MinDom}(T.\text{root}, S, m) \)
5: \( \hat{R}(S) \leftarrow \hat{D}(T.\text{root}, S) + 1 \) \text{ // ranking lower bound of missing object \( m \) under keyword set \( S \)}
6: \( \check{R}(S) \leftarrow \check{D}(T.\text{root}, S) + 1 \) \text{ // ranking upper bound of \( m \) under \( S \)}
7: insert \( T.\text{root} \) into \( Q \)
8: \textbf{while} \( Q \) is not empty \textbf{do}
9: \( N \leftarrow \text{Dequeue}(Q) \)
10: \textbf{for} each keyword set \( S \) in \( CK \) \textbf{do}
11: \( \hat{D}'(N, S) \leftarrow 0 \)
12: \( \check{D}'(N, S) \leftarrow 0 \)
13: \textbf{for} each child \( c \) of \( N \) \textbf{do}
14: \( \hat{D}(c, S) \leftarrow \text{MaxDom}(c, S, m) \)
15: \( \check{D}(c, S) \leftarrow \text{MinDom}(c, S, m) \)
16: \( \hat{D}'(N, S) \leftarrow \hat{D}'(N, S) + \hat{D}(c, S) \)
17: \( \check{D}'(N, S) \leftarrow \check{D}'(N, S) + \check{D}(c, S) \)
18: \( \hat{R}(S) \leftarrow \hat{R}(S) - (\hat{D}(N, S) - \hat{D}'(N, S)) \)
19: \( \check{R}(S) \leftarrow \check{R}(S) - (\check{D}(N, S) - \check{D}'(N, S)) \)
20: compute \( \hat{p}(S), \check{p}(S) \) from \( \hat{R}(S), \check{R}(S) \) according to Eqn. (4.1.4)
21: \textbf{if} \( \hat{p}(S) < p_c \) \textbf{then}
22: \( p_c \leftarrow \hat{p}(S) \)
23: \( \text{doc}' \leftarrow S \)
24: \textbf{for} each keyword set \( S \) in \( CK \) \textbf{do}
25: \textbf{if} \( \check{p}(S) > p_c \) \textbf{then}
26: prune \( S \) from \( CK \)
27: \textbf{if} \( S \) is empty \textbf{then}
28: \textbf{return} \( (loc, \text{doc}', k', \alpha), p_c \)
29: \textbf{for} each child \( c \) of \( N \) \textbf{do}
30: \textbf{if} \( c \) is an object or \( \hat{D}(c, S) = \check{D}(c, S) \) for all \( S \) in \( CK \) \textbf{then}
31: \textbf{continue}
32: insert \( c \) into \( Q \)
33: \textbf{return} \( (loc, \text{doc}', \check{R}(\text{doc}'), \alpha), p_c \) \text{ // \( \check{R}(\text{doc}') = \check{R}(\text{doc}') \) at last}

sets. While the straightforward way of using the algorithm may work well for a small number of candidate keyword sets, we proceed to present a strategy to divide all the candidates into subsets according to their penalty, \( i.e., \) the edit distance, from the initial query keyword set. This helps speed up the process and triggers early stop
that avoids enumerating all candidates. We access the subsets in ascending order of their penalty, and for each such subset, we invoke Algorithm 5 to determine the best refined query. The process stops when the penalty of the known best refined query is no larger than that of the next retrieved subset.

The details are shown in Algorithm 6. The first step is to determine the ranking of the missing object under the initial query, i.e., $R(m, q)$ (Line 1). This can be done by slightly modifying the underlying spatial-keyword top-$k$ algorithm by changing the stop condition from retrieving top-$k$ objects to retrieving the missing object $m$. Then we initialize the currently best refined query to be the basic one (Line 2). Afterwards, we iteratively find the subset of query keywords in ascending order of their edit distance to the initial query keywords and invoke Algorithm 5 until the next subset’s penalty in the textual dimension is no less than that of the known best refined query (Lines 3–7).

**Algorithm 6** Answering Why-not Query

**INPUT:** KcR-tree $\mathcal{T}$, original query $q = (\text{loc}, \text{doc}_0, k_0, \alpha)$, missing object $m$

**OUTPUT:** Best refined query $q' = (\text{loc}, \text{doc}', k', \alpha)$

1: determine $R(m, q)$

2: $\text{doc}' \leftarrow \text{doc}_0, k' \leftarrow R(m, q), p_c \leftarrow \lambda$ // initialize the best refined query and the penalty threshold with the very basic refined query

3: for $k$ from 1 to $|\text{doc}_0 \cup m.\text{doc}|$ do

4: if $(1 - \lambda) \cdot \frac{k}{|\text{doc}_0 \cup m.\text{doc}|} \geq p_c$ then

5: break

6: $\text{CK} \leftarrow \text{NextKeywordSets}()$ // find the next set of keyword sets that has $k$ edit distance from $\text{doc}_0$

7: invoke Algorithm 5 using $(\mathcal{T}, q', \text{CK}, p_c, m)$ to determine the currently best refined query $q'$ and its penalty $p_c$

8: return $q' = (\text{loc}, \text{doc}', k', \alpha)$
4.4 Multiple Missing Objects and Approximate Algorithm

4.4.1 Multiple Missing Objects

Both proposed algorithms can be extended to handle queries with multiple missing objects. In particular, two issues must be addressed to contend with multiple missing objects. The first is to find those candidate keyword sets to consider. The second is to include all missing objects when applying the algorithms.

Regarding the first issue, recall that the candidate keyword sets for the refined query is obtained from modifying the initial query keywords $doc_0$ by adding keywords to $doc_0$ and/or deleting existing keywords from $doc_0$. In queries with multiple missing objects, we consider adding only the keywords in $M.doc$, where $M.doc = \bigcup_{i=1}^{j} m_i.doc$. There are two reasons. First, adding a keyword that is not in $M.doc$ makes the set of query keywords less relevant to the user’s query intention, i.e., less relevant to any of the missing objects. Second, if we were to consider adding a keyword $t$ not in $M.doc$, it is best to add a keyword that is not even in the whole dataset, as such a keyword does not also increase any other object’s textual similarity. However, this would also make the refined queries less relevant to the missing objects.

To achieve the inclusion of all missing objects, we slightly modify the algorithms. First, we use $R(M, q)$ instead of $R(m, q)$ when estimating the penalty (Eqn. (4.1.4)). In the basic algorithm, the largest modification is to stop a generated spatial keyword query when all missing objects are retrieved. Similarly, in the KcR-tree based algorithm, for each candidate keyword set, $MaxDom(T, S, M)$ and $MinDom(T, S, M)$ are estimated as $\max_{m_i \in M} MaxDom(T, S, m_i)$ and $\min_{m_i \in M} MinDom(T, S, m_i)$, respectively. The estimation of the missing objects’ rankings ($\hat{R}(S)$ and $\check{R}(S)$) and the penalty ($\hat{pn}(S)$ and $\check{pn}(S)$) w.r.t. each candidate keyword set $S$ is changed accordingly. All the optimization techniques can be adapted similarly to support queries with multiple missing objects.
4.4.2 Approximate Algorithm

So far, we have focused on finding the exact solution with the least penalty among all candidate keyword sets. The number of candidate keyword sets grows fast when \(|doc_0 \cup M.doc|\) increases. Although several optimizations and an index-based algorithm are proposed for the why-not question, the exact algorithm could still be costly when the number of the query keywords is huge. In such cases, we can trade solution quality for execution time. Instead of taking into consideration of all the candidate keyword sets, we apply the algorithms only to a sample of all sets to find an approximate solution. As with typical sampling-based methods [17, 19], two key issues need to be addressed: the sample size and how to obtain high quality keyword sets in the sample.

A larger sample size is more likely to yield a higher quality solution. However, a larger sample size will also result in longer processing time. A nice property of the proposed algorithms is that they can work on samples of any size, which lends themselves naturally to enable a tradeoff between result quality and running time. We shall study the effect of the sample size \(T\) experimentally in Section 4.5.2.

To sample high quality keyword sets, based on the analysis of the enumeration order that we discussed in Section 4.2.3, we greedily choose the first \(T\) keyword sets with the highest total particularity w.r.t. the missing objects due to editing the initial query keyword set.

4.5 Empirical Study

The ensuing experimental study primarily considers three methods: the basic algorithm developed in Section 4.2.2 (referred to as \(BS\)), the basic algorithm with the optimizations from Section 4.2.3 (referred to as \(AdvancedBS\)), and the KcR-tree approach with the optimizations developed in Section 4.3 (referred to as \(KcR-Based\)). We also implement the approximate algorithm and evaluate its performance and solution quality.
### 4.5.1 Experimental Setup

**System Setup and Metrics**

All experiments are conducted on a PC with an Intel Core i7 3.4GHz CPU and 16GB memory running Windows 7 OS. The algorithms are implemented in Java, and the maximum main memory of the Java Virtual Machine is set to 4GB. The index structures, the \textit{SetR-tree} and the \textit{KcR-tree}, are both disk-resident. The page size is set to 4KB, the buffer size is set to 4MB, and the capacity of a node is set to 100. For all algorithms under evaluation, we use two metrics, the number of I/Os and the query time, to evaluate their performance. For each experiment, we randomly generate 1,000 queries and report the average result.

**Datasets**

We use two real datasets, EURO and GN, in the experiments. EURO is a dataset of points of interest like ATMs, hotels, and stores in Europe (www.allstays.com); and GN is obtained from the US Board on Geographic Names (geonames.usgs.gov) and contains a set of geographic objects. Both of them are commonly used in spatial keyword related research [4, 5, 29, 39]. Each dataset contains a number of objects represented by a spatial location and a set of keywords. More details about the datasets are provided in Table 4.2.

**Parameters**

We evaluate the performance of our algorithms by varying different parameters. The parameters together with their default values (in bold face) are shown in Table 4.3.

### Table 4.2: Dataset Information

<table>
<thead>
<tr>
<th>Dataset</th>
<th>EURO</th>
<th>GN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total # of objects</td>
<td>162,033</td>
<td>1,868,821</td>
</tr>
<tr>
<td>Total # of distinct words</td>
<td>35,315</td>
<td>222,407</td>
</tr>
</tbody>
</table>
Unless specified otherwise, the experiments are conducted using the dataset EURO, and we set the missing object as the one ranked at $5 \cdot k_0 + 1$ in the initial query.

Table 4.3: Parameter Setting

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_0$</td>
<td>3, 10, 30, 100</td>
</tr>
<tr>
<td># of keywords</td>
<td>2, 4, 6, 8</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.1, 0.3, 0.5, 0.7, 0.9</td>
</tr>
<tr>
<td>$R(m, q)$</td>
<td>31, 51, 101, 151, 201</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.1, 0.3, 0.5, 0.7, 0.9</td>
</tr>
<tr>
<td># of missing objects</td>
<td>1, 2, 3, 4</td>
</tr>
</tbody>
</table>

4.5.2 Experimental Result

Varying $k_0$

We first vary parameter $k_0$ in the initial query to observe its effect on the performance of the algorithms. The ranking of the missing object varies along with $k_0$, i.e., $R(m, q) = 5 \cdot k_0 + 1$. For instance, when the initial query is varied from a top-3 to a top-10 query, corresponding why-not queries are posed to identify keyword sets that retrieve objects that rank from 16 to 51. Fig. 4.4 shows the results. Recall that in the basic algorithm, a spatial keyword query is executed for each candidate keyword set until the missing object is retrieved. Since the ranking of the missing object drops as $k_0$ increases, the time for executing a spatial keyword query increases. Thus, the basic algorithm is quite sensitive to changes in $k_0$. On the other hand, thanks to the optimization techniques developed, AdvancedBS and KcRBased scale much better when $k_0$ increases, and KcRBased achieves the best performance. For example, when $k_0 = 100$, KcRBased runs almost 5 times faster than BS and reduces the I/O by more than 70%.
Varying the number of initial query keywords

This set of experiments evaluates the effect of varying the number of query keywords in the initial query. The results are shown in Fig. 4.5. Intuitively, the number of keywords influences the performance of the algorithms in two aspects. First, more query keywords mean that more time will be consumed to compute the textual similarity between tree nodes/objects and query keywords; second, the candidate query keyword sets may grow exponentially when the number of initial query keywords increases.

The number of candidate keyword sets is a dominant factor. As the basic algorithm needs to process a spatial keyword query to determine the rank of the missing object under each candidate keyword set, the query time of BS increases dramatically when the number of initial query keywords increases. In contrast, AdvancedBS and KcRBased increasingly outperform BS when the number of initial query keywords increases. For queries with 6 or more query keywords, KcR-Based runs 1.5 times faster than AdvancedBS and outperforms BS by an order of magnitude in query time.

Varying $\alpha$

In this set of experiments, we study the effect of parameter $\alpha$. Fig. 5.10 plots the results. According to the ranking function in the spatial keyword top-$k$ query, i.e.,
Eqn. (4.1.1), a smaller $\alpha$ means a higher weight to the textual similarity, which reduces the importance of spatial distance. As a result, the pruning ability of the R-tree based indexes decreases, and more tree nodes may need to be accessed. That is the main reason why a smaller $\alpha$ causes more I/O in all tested algorithms. On the other hand, as a large (resp., small) $\alpha$ gives higher weight to the spatial (resp., textual) dimension in the ranking function, this reduces the pruning capability in the other dimension. This may be the reason why a medium $\alpha$ has the least execution time.
Varying $\lambda$

Next we investigate the effect of parameter $\lambda$ in the penalty function, which allows users to indicate their preferences on modifying the query keywords versus modifying $k$. As shown in Fig. 4.7, the basic algorithm is almost unaffected by $\lambda$. The reason is that in $BS$, $\lambda$ is only used to compute the penalty of a candidate keyword set after the generated spatial keyword query determines the rank of the missing object; the computation is exactly the same for different $\lambda$ values. However, in both $AdvancedBS$ and $KcRBased$, the currently best refined query and its penalty are maintained for further pruning, which is initialized using the basic refined query that keeps the query keywords and sets $k_0$ to $R(m, q)$ to include the missing object. According to Eqn. (4.1.4), the penalty of the basic refined query is $\lambda$. A smaller $\lambda$ leads to a smaller initially seen penalty in $AdvancedBS$ and $KcRBased$, which can improve the pruning ability. Therefore, the query time of $AdvancedBS$ and $KcRBased$ increases with $\lambda$. However, $KcRBased$ is more stable than $AdvancedBS$.

![Figure 4.7: Varying $\lambda$](image)

(a) Query Time 
(b) Page Access 

Varying the rank of the missing object

In this set of experiments, we study the performance when the initial rank of the missing object is varied. Since the initial query is a spatial keyword top-10 query, we ask five different why-not questions with the missing object ranked 31, 51, 101, 151, 

70
and 201. As shown in Fig. 4.8, the performance of BS is much more sensitive to changes in the rank of the missing object, whereas AdvancedBS and KcRBased are affected only slightly. The reason is the same as when varying $k_0$. In fact, the results of this set of experiments and those of varying $k_0$, i.e., Fig. 4.8 and Fig. 4.4, are quite similar. This implies that the performance of the algorithms is affected significantly by the initial rank of the missing object and has little to do with $k_0$.

![Graphs showing query time and page access for varying initial rankings](image)

Figure 4.8: Varying the missing object’s initial ranking

Varying the number of missing objects

We also study performance when the number of the missing objects changes. In this set of experiments, the initial query is a top-10 spatial keyword query with 4 query keywords. The missing objects are randomly selected from the objects ranked between 11 and 51 w.r.t. the initial query. Fig. 4.9 plots the results. We can observe that the number of missing objects has a remarkable effect on the performance of the algorithms. The reason is that for multiple missing objects, we need to consider the union of all the missing objects’ keywords as the search space, which means that the number of the candidate keyword sets may increase dramatically along with the increasing number of missing objects. We can also see that AdvancedBS and KcRBased scale much better than BS when the number of missing objects grows.
Varying the number of threads

In Fig. 4.10, we further evaluate the performance of the algorithms when the spatial keyword queries are processed in parallel. The \textbf{KcRBased} algorithm is parallelized by dividing the set of candidate keyword sets into several smaller sets and running the algorithm on each set while the currently known best refined query and its penalty are synchronized for pruning and early termination. The performance of the algorithms can be accelerated significantly by using up to 8 threads.

Pruning abilities of optimizations

We show the query performance of different optimizations in Fig. 4.11, where Opt1, Opt2, and Opt3 represent the strategies of early stopping, considering the enumer-
ation order, and keyword set filtering, respectively. Each optimization reduces the query time. In particular, keyword set filtering is the most effective, as it prunes candidate keyword sets before even processing the corresponding spatial keyword queries.

Approximate Algorithm

In this set of experiments, we evaluate the performance of the approximate algorithm (Section 4.4.2). The initial query is a top-10 query with 8 query keywords. The approximate algorithm is implemented by sampling different numbers, between 100 and 800, of keyword sets from all the candidate sets. The query time and the average penalty of the returned best refined query w.r.t. the sample size together with those of the exact algorithm are shown in Fig. 4.12. We can see that the query time of \( BS \) is linear to the sample size. This is because, for each sample, a spatial keyword query needs to be processed until retrieving the missing object. Moreover, for each sample size, the penalties of the refined queries are the same when using the different algorithms. This is because the sample space is the same and each algorithm returns the best refined query among the samples. We can also observe that the penalty generally decreases as the sample size increases. In particular, for \( KcRBased \), when the sample size is 800, the approximate algorithm sacrifices only 12% of the penalty while saving 30% of the query time.
Finally, we test the scalability of the algorithms. We randomly select different numbers of objects from the GN dataset to evaluate the query performance under different dataset sizes. The initial spatial keyword query is a top-10 query. Fig. 4.13 plots the result. As we can see, the query time and page access of the algorithms grow almost linearly when the dataset cardinality increases. Since we do not change the candidate keyword sets with the increase of the dataset size, the number of spatial keyword queries varies only little. The cost of processing a spatial keyword query increases linearly with the dataset size, which then explains the performance trend of our algorithms under different dataset sizes.

4.6 Summary

In this chapter, we have studied the problem of answering why-not queries in the context of spatial keyword top-$k$ queries by refining the original query keywords, which provides users with keywords that better describe their query intention. We have proposed a basic algorithm with a set of optimization techniques that finds the best solution based on testing the candidate keyword sets one by one. Furthermore, we have proposed a more efficient KcR-tree-based algorithm that quickly determines the best solution among all candidates using a bound-and-prune strategy. We have
also extended these algorithms to handle multiple missing objects and presented a sampling-based approximate algorithm. Extensive experiments on real datasets demonstrate that the optimized algorithm and the KcR-tree-based algorithm are scalable and able to reduce the query time by up to an order of magnitude in various settings. In addition, the approximate algorithm achieves a good tradeoff between result quality and running time.

This chapter is a slightly amended version of the paper [9] that we published in ICDE 2016. The DOI of the paper is 10.1109/ICDE.2016.7498282.
Chapter 5

Direction-Aware Why-Not Spatial Keyword Top-$k$ Query

It is relevant to take into account the query direction in spatial keyword queries in a number of scenarios. For example, a user walking to a supermarket may want to find an ATM in her walking direction. As another example, a user on a high-way may want to find a gas station or restaurant in the users general travel direction (i.e., the right front region in right-driving countries). Considering that it is difficult for users to specify proper query directions, we study the problem of direction-aware why-not spatial keyword top-$k$ queries in this chapter, which is able to provide users more accurate query directions that better describe the intents of their queries. This chapter is organized as follows. Section 5.1 introduces preliminaries and defines the direction-aware why-not spatial keyword query problem. Sections 5.2 and 5.3 present the problem analysis and solutions in the two different cases. We extend the algorithms to support queries with multiple missing objects in Section 5.4. The experimental studies are covered in Section 5.5. Finally, we conclude in Section 5.6.

5.1 Preliminaries and Problem Formulation

In this section, we first review the top-$k$ and direction-aware top-$k$ spatial keyword queries, and then we formalize the problem of direction-aware why-not spatial key-
5.1.1 Preliminaries

Let \( D \) denote a database of spatial objects. Each object \( o \in D \) is associated with a pair \((o.loc, o.doc)\), where \( o.loc \) is the object’s spatial location and \( o.doc \) is a set of keywords that describes the object.

A spatial keyword top-\( k \) query \( q \) takes four parameters \((loc, doc, \vec{w}, k)\). Here \( q.loc \) is the query location, \( q.doc \) is a set of query keywords, \( q.k \) denotes the number of objects to retrieve, and \( q.\vec{w} = (w_s, w_t) \), where \( 0 \leq w_s, w_t \leq 1 \) and \( w_s + w_t = 1 \), denotes the user’s preferences between spatial proximity and textual relevance. The query retrieves the top-\( k \) objects from \( D \) ranked according to a scoring function that aggregates the spatial distance and textual similarity into an overall scoring value. For broad applicability, we adopt a widely used ranking function [14]:

\[
ST(o, q) = w_s \cdot (1 - SDist(o, q)) + w_t \cdot TSim(o, q),
\]

(5.1.1)

where \( SDist(o, q) \) denotes the Euclidean distance between \( o.loc \) and \( q.loc \), and \( TSim(o, q) \) denotes the textual similarity between \( o.doc \) and \( q.doc \). The spatial distance \( SDist(o, q) \) is normalized into the range \([0, 1]\) by dividing the maximum possible distance between two objects in \( D \). The textual similarity \( TSim(o, q) \) can be computed using an information retrieval model [30], such as the language model, cosine similarity, Jaccard similarity, or BM25, and is also assumed to be normalized into the range \([0, 1]\). Without loss of generality, we adopt the language model [14].

Given a query \( q \), the rank of an object \( o \) is given in terms of Eqn. (5.1.1) as follows:

\[
R(o, q) = |\{o' \in D \mid ST(o', q) > ST(o, q)\}| + 1
\]

(5.1.2)

A spatial keyword top-\( k \) query \( q \) returns a set \( R \) of \( k \) objects from \( D \), where

\[
\forall o \in R \ (\forall o' \in D - R \ (ST(o, q) \geq ST(o', q))),
\]

or in terms of object ranking,

\[
\forall o \in R \ (\forall o' \in D - R \ (R(o, q) \leq R(o', q))).
\]

While the spatial keyword top-\( k \) query has been studied extensively, direction-aware search, which is useful in many LBS scenarios [25], has received only little
attention. In a direction-aware spatial keyword top-k query, an object can be a result only if it is located in a certain direction of a query location. A direction is defined in terms of rays emanating from the query location [25]. Without loss of generality, we assume that the objects are mapped to a Cartesian coordinate system. We delineate a direction by the angles between two rays and the positive direction of the x-axis. The direction $d$ in a query is a range $(\alpha, \beta)$, denoting that the query is interested only in objects in the direction $(\alpha, \beta)$. That is, a query’s direction is an angular space. As such, a direction-aware spatial keyword top-k query $q_d$ is a 5-tuple $(\text{loc}, \text{doc}, \bar{w}, k, d)$.

**Definition 5.1.1. Direction-Aware Spatial Keyword Top-k Query.** Let $D_d$ denote the objects in $D$ that are located in the angular region $d = (\alpha, \beta)$, and let $R(o, q, d)$ denote the rank of an object $o$ under a query $q_d$. A direction-aware spatial keyword top-k query $q_d$ returns a set $\mathcal{R}$ of $k$ objects from $D_d$, where $\forall o \in \mathcal{R}$ ($\forall o' \in D_d - \mathcal{R}$ ($R(o, q, d) \leq R(o', q, d)$)).

In other words, instead of considering the whole database $D$, a direction-aware spatial keyword query considers only the objects in a directional range as candidates for the query result. We note that the traditional spatial keyword top-k query can be treated as a direction-aware query with a direction of $[0, 2\pi)$.

**Example 8.** Fig. 5.1 shows an example of the top-k and the direction-aware top-k spatial keyword queries, where (a) shows the locations of the query and objects, while (b) lists the ranking score of each object. Consider a top-2 query. A traditional query returns the objects with the highest scores among all the objects, i.e., $o_6$ and $o_1$. However, if the query has a direction, say, $(\frac{\pi}{6}, \frac{\pi}{3})$, the query considers only the objects in this direction and returns the top-2 objects among them, i.e., $o_1$ and $o_3$.

\[1\] A direction $(\alpha, \beta)$ is the angular space passed through by rotation from $\alpha$ to $\beta$ counterclockwise. We use open or closed intervals to denote whether a boundary direction is included. For the sake of presentation, we convert all directions to the space where $-\pi < \alpha \leq \pi$ and $\alpha \leq \beta \leq \alpha + 2\pi$. We say a direction $\theta \in (\alpha, \beta)$, if $\exists n \in \mathbb{Z}$, $\theta + 2n\pi \in (\alpha, \beta)$. 

78
5.1.2 Direction-Aware Why-Not Spatial Keyword Query

When issuing a direction-aware spatial keyword query \( q = \{\text{loc}, \text{doc}, \bar{w}, k_0, d_0\} \), it might be difficult for a user to specify a direction \( d_0 \) that best captures the query intent. Due to an improper setting of the query direction, after the user receives a query result, the user may find that one or more desired objects are unexpectedly missing. These missing objects imply the user’s actual direction requirement. Then, the user may pose a follow-up why-not question with a set \( M = \{m_1, m_2, ..., m_j\} \) of desired but missing objects, asking why these expected objects are missing and seeking a refined query \( q' = \{\text{loc}, \text{doc}, \bar{w}, k', d'\} \) that includes the missing objects in its result. As simply modifying the direction in the initial query may not be able to revive the missing objects, the enlargement of \( k \) is also considered. Many possible modifications of these two parameters may yield a qualified query retrieving the missing objects. We prefer the one that modifies the initial query minimally. To formalize this, we adopt a penalty model [17, 19] that quantifies the modification as a weighted sum of the changes of parameters \( \Delta k \) and \( \Delta d \). The penalty of a refined query \( q' \) against the initial query \( q \) is defined as follows:

\[
\text{Penalty}(q, q') = \lambda \cdot \frac{\Delta k}{\Delta k_{\text{max}}} + (1 - \lambda) \cdot \frac{\Delta d}{\Delta d_{\text{max}}},
\]

(5.1.3)

where \( \lambda \in [0, 1] \) is a user preference on modifying \( k \) versus \( d \). Here, \( \Delta k_{\text{max}} \) and \( \Delta d_{\text{max}} \) denote the maximum possible modifications of \( k \) and \( d \), respectively. They are used to normalize \( \Delta k \) and \( \Delta d \) into the range \([0, 1]\). Since their settings would
vary in different cases, we leave their definitions to the coverage of the corresponding
cases in Sections 5.2 and 5.3. We have \( \Delta k = \max(0, k' - k_0) \), since \( k' \) can remain
as \( k_0 \) if a refined \( k' \) is smaller than \( k_0 \). As the query direction \( d \) is an angular space
between the start angle \( \alpha \) and the end angle \( \beta \), we measure the modification from
\( d_0 = (\alpha_0, \beta_0) \) to \( d' = (\alpha', \beta') \) in terms of how much \( d_0 \) is rotated and how much the
size of \( d_0 \) is changed, \( i.e., \Delta r \) and \( \Delta s \). Formally, \( \Delta d \) is defined as follows:
\[
\Delta d = \gamma \cdot \Delta r + (1 - \gamma) \cdot \Delta s
\]
\[
= \gamma \cdot \left| \frac{\alpha' + \beta'}{2} - \frac{\alpha_0 + \beta_0}{2} \right|
+ (1 - \gamma) \cdot |(\beta' - \alpha') - (\beta_0 - \alpha_0)|
\]
(5.1.4)
The rotation of the direction \( \Delta r \) is determined by the difference between the angular
bisectors, \( i.e., \frac{\alpha' + \beta'}{2} \) and \( \frac{\alpha_0 + \beta_0}{2} \). Finally, \( \gamma \in [0, 1] \) is used to balance the changes to
the rotation and the size of the direction.

Based on the above, the direction-aware why-not spatial keyword query is defined
as follows:

**Definition 5.1.2. Direction-Aware Why-Not Spatial Keyword Top-\( k \) Query.**

Given a set \( D \) of spatial objects, a missing object set \( M \), an original direction-aware
spatial keyword query \( q = (\text{loc}, \text{doc}, \tilde{w}, k_0, d_0) \), the direction-aware why-not spatial
keyword top-\( k \) query returns the refined query \( q' = (\text{loc}, \text{doc}, \tilde{w}, k', d') \), with the low-
est penalty according to Eqn. (5.1.3) and the result of which includes all objects in
\( M \).

5.1.3 Baseline Algorithm

We consider refining the direction \( d_0 \) and the result cardinality \( k_0 \) to achieve the
inclusion of the missing objects. Only refined pairs \((k', d')\) that satisfy Lemma 5.1.3
are candidates for the best refined query.

**Lemma 5.1.3.** Given an initial query \( q \) and a set \( M \) of missing objects, a pair of
a modified direction and a refined result cardinality \((d', k')\) can be a candidate for
the best refined query if and only if \((i) \ \forall m_i \in M \ (\theta_{m_i} \in d')\), where \( \theta_{m_i} \) denotes
Proof. The proof is straightforward and hence omitted.

According to Lemma 5.1.3, given a refined direction $d'$, we can always set $k' = R(M, q, d')$ to achieve the minimum penalty. In other words, if we fix parameter $d'$, $k'$ can be set accordingly. This observation inspires a baseline solution as follows: (i) enumerate all possible refined directions; (ii) for each candidate direction, process a direction-aware spatial keyword top-$k$ query to determine the ranks of the missing objects; (iii) compute the penalty of each candidate direction and return the one with the minimum penalty.

Two key challenges exist in this baseline solution. First, the number of possible directions is generally infinite, making enumeration impossible. One way to overcome this issue is to sample part of the candidate refined directions [17, 19]. Nevertheless, it is hard to guarantee the solution quality, and the baseline may fail to find the optimal solution. Second, the baseline needs to invoke a spatial-keyword query for each enumerated direction, where high computation and I/O costs are incurred. As such, the baseline algorithm might be inefficient and inapplicable to the general case. In the following sections, we develop more efficient solutions based on a careful problem analysis. We aim to invoke the spatial keyword query only once during the why-not query processing. We consider a single missing object in Sections 5.2 and 5.3, and we extend the algorithms to support multiple missing objects in Section 5.4.

5.2 Answering Why-Not Questions: Special Case

In this section, we assume a single missing object and study the direction-aware why-not problem for the special case, where the initial query is a traditional query with no query direction. This case may occur when users do not at first realize their query direction requirements; or they do not indicate a query direction, possibly because
they would like the system to add the direction as a direction is more difficult to specify than it is to point out some expected result object(s).

5.2.1 Case Analysis

Recall that the traditional spatial keyword top-k query can be treated as a query with a direction of \([0, 2\pi]\). Actually, it can be represented by a direction-aware query with a direction in \(\{d \mid d = [\alpha, \alpha + 2\pi), -\pi < \alpha \leq \pi\}\). In other words, the bisector of the direction can be any ray around the query location. Therefore, for a refined direction \((\alpha', \beta')\), we can always find an \(\alpha_0\) that makes \(\frac{\alpha' + \beta'}{2} = \frac{\alpha_0 + \beta_0}{2}\). Thus, the modification in rotating the initial direction can be treated as 0, and \(\Delta d\) can be rewritten as follows:

\[
\Delta d = 2\pi - (\beta' - \alpha')
\]  

(5.2.5)

Thus, the maximum possible modification of the direction is \(\Delta d_{max} = 2\pi\), which is obtained when \(\alpha' = \beta'\). Moreover, \(\Delta k_{max}\) can be estimated as \(R(m, q, d_0) - k_0\), where \(R(m, q, d_0)\) denotes the rank of the missing object \(m\) under the initial query, as a very naive method to revive the missing object is to increase \(k_0\) until \(m\) is included in the result without adjusting the direction. As such, the penalty function becomes:

\[
Penalty(q, q') = \lambda \cdot \frac{\Delta k}{R(m, q, d_0) - k_0} + (1 - \lambda) \cdot \frac{2\pi - (\beta' - \alpha')}{2\pi}
\]

(5.2.6)

To achieve the minimum penalty, for a given \(k'\), the largest direction that could rank the missing object within top-\(k'\) is preferred.

5.2.2 Ranking Updates and Direction Modification

To answer the direction-aware why-not query, we make two observations. First, after the initial query is issued, the ranking score of each object is a constant. Second, the query direction works as a filter, where only the objects in the query direction are candidates for the result. Therefore, the rank of the missing object in the refined query is determined by the number of objects in the refined query direction that
ranks better. As mentioned, the query direction $d$ of the spatial keyword query has a start angle $\alpha$ and an end angle $\beta$. To determine a refined direction, we need to identify its start and end angles. Given a start angle $\alpha'$ of a refined direction, the following theorem holds.

**Theorem 5.2.1.** Consider an initial query $q$, a missing object $m$, and a refined start angle $\alpha'$. Let $\theta_m$ denote the angle of $m$ w.r.t. $q$. If a direction $(\alpha', \beta')$ is a candidate for the best refined query, it holds that: (i) $\theta_m \in (\alpha', \beta')$; and (ii) there exists an object $o$ with angle $\beta'$ such that $ST(o, q) > ST(m, q)$.

*Proof.* The first condition is obviously necessary. It ensures that the missing object is in the query direction. We prove the second condition by contradiction. Assume that $\beta'$ is a candidate end angle and that no object with angle $\beta'$ scores higher than the missing object $m$ under the initial query $q$. Then there are only two cases: there exists objects that rank higher than $m$ in $(\beta', \alpha' + 2\pi]$; or no object ranks higher than $m$ in $(\beta', \alpha' + 2\pi]$. In the former case, let $o'$ be the first object that ranks higher than $m$ in $(\beta', \alpha' + 2\pi]$, and let $\theta_{o'}$ denote its angle w.r.t. $q$. Let $\beta''$ be any angle in $(\beta', \theta_{o'})$. Since $\forall o \in (\beta', \theta_{o'}) (ST(o, q) \leq ST(m, q))$, the rank of the missing object would be the same in the directions $(\alpha', \beta')$ and $(\alpha', \beta'')$. Thus, $\Delta k$ is identical for the end angles $\beta'$ and $\beta''$. However, as $(\alpha', \beta') \subset (\alpha', \beta'')$, $\Delta d_{\beta'} > \Delta d_{\beta''}$. According to Eqn. (5.2.6), for a given $k'$, a larger direction is preferred. The penalty for the end angle $\beta'$ exceeds that of $\beta''$. Thus, $\beta'$ cannot be a candidate end angle for start angle $\alpha'$, which contradicts our assumption. Similarly, in the latter case, any angle in $(\beta', \alpha' + 2\pi]$ would have a smaller penalty than $\beta'$ so that $\beta'$ cannot be a candidate end angle. Thus, Theorem 5.2.1 holds. $\square$

Similarly, for a given refined end angle $\beta'$, the refined start angle $\alpha'$ should be set to $\beta' - 2\pi$, or should be the angle of an object that ranks higher than the missing object under the initial query.

**Example 9.** In Fig. 5.2, the objects that have ranking scores higher than the missing object $m$ under the initial query are marked as black points and the others are marked
Figure 5.2: A candidate end (start) direction for a given $\alpha'$ ($\beta'$) in grey. In Fig. 5.2(a), for start angle $\alpha'$, when increasing the end angle $\beta'$ from $\beta_1$ to $\beta_2$, the rank of $m$ in the direction $($$\alpha'$, $\beta'$$)$ remains unchanged. But as the size of the direction increases, according to Eqn. (5.2.6), the penalty for the refined direction $($$\alpha'$, $\beta'$$)$ keeps decreasing. Consequently, no end angle between $\beta_1$ and $\beta_2$ can yield the best refined direction. The case is similar for the candidate start angles when a refined end angle $\beta'$ is given (See Fig. 5.2(b)).

These observations yield the following proposition for the candidate refined directions:

**Proposition 5.2.2.** A refined direction $($$\alpha'$, $\beta'$$)$ can result in the best refined query if: (i) $\theta_m \in ($$\alpha'$, $\beta'$$)$; and (ii) both the start angle $\alpha'$ and the end angle $\beta'$ have an object that ranks higher than the missing object under the initial query.

Proposition 5.2.2 limits the candidate directions to a finite set and thus enables enumeration. We also note that the rank of the missing object $m$ in a direction $($$\alpha'$, $\beta'$$)$ is determined by the number of objects that have higher ranking scores than $m$, hereafter called $m$'s *dominators*, in the direction. These dominator objects are determined when the initial query is issued. Thus, instead of processing a spatial keyword query to compute the rank of the missing object for each candidate direction, we can first identify $m$'s dominators and then determine $m$'s rank by counting how many of them are in each candidate direction.
5.2.3 Answering Why-Not

Based on the above discussions, we present the proposed algorithm for answering direction-aware why-not questions in the special case. The pseudo-code is given in Algorithm 7. We first compute the rank of the missing object under the initial query, i.e., $R(m, q, d_0)$, and we record $m$’s dominators (Line 1). This can be done by slightly modifying the underlying spatial keyword top-$k$ algorithm (e.g., [37]) by changing the stop condition from retrieving $k$ objects to retrieving the missing object. We then invoke the function $CalDirection$ to calculate the angles of the missing object and its dominators w.r.t. the initial query $q$ (Lines 2–4). The function $CalDirection$ confines the angles of the objects within the range $(-\pi, \pi]$. For ease of presentation, we assume no $m$’s dominators locate at the query location. The algorithm can be adapted easily to support this case by treating all such dominators to be in any direction. Next, $m$’s dominators are sorted in clockwise order according to their angles w.r.t. the missing object’s angle, i.e., in ascending order of $(\theta_m - \theta[i] + 2\pi) \% 2\pi$ (Line 5).\footnote{We use $\%$ to denote the modular operation throughout this chapter.} Afterwards, we initialize the currently seen best refined query with the basic one that simply modifies $k_0$ to $R(m, q, d_0)$ (Line 6). We then enumerate each possible refined $k'$ in increasing order (Line 7). At $k'$, if merely modifying $k_0$ to $k'$ results in a penalty larger than the minimum obtained so far, the process is terminated (Lines 8–10). The range of a possibly refined $k'$ is from $k_0$ to $R(m, q, d_0) - 1$, since for $k' < k_0$, the candidate directions for $k'$ would be contained in that of $k_0$ and hence cannot achieve a smaller penalty. For each possibly refined $k'$, we check the candidate directions that rank $m$ as a top-$k'$ object and compute their penalties to examine whether they are better than the currently seen best refined query (Lines 11–16).

Example 10. Take Fig. 5.3 as an example. Assume that the initial query $q$ is a traditional spatial keyword top-3 query. Six objects dominate $m$ w.r.t. $q$, i.e., $R(m, q, d_0) = 7$. This makes $m$ missing from the top-3 result. We first identify these $m$’s dominators and compute their angles. To easily locate the candidate
Algorithm 7 Answering Why-Not Questions: Special Case

INPUT: Original query $q = (\text{loc}, \text{doc}, \vec{w}, k_0, d_0 = [-\pi, \pi])$, Missing object $m$

OUTPUT: Best refined query $q' = (\text{loc}, \text{doc}, \vec{w}, k', d')$

1: compute $R(m, q, d_0)$ and record $m$’s dominators in set $S$
2: $\theta_m \leftarrow \text{CalDirection}(q, m)$
3: for each $a_i \in S$
4: \hspace{1em} $\theta[i] \leftarrow \text{CalDirection}(q, a_i)$
5: sort $\theta[i]$ in clockwise order w.r.t. $\theta_m$
6: $d' \leftarrow d_0$, $k' \leftarrow R(m, q, d_0)$, $p_c \leftarrow \lambda$
7: for $r = k_0$ to $R(m, q, d_0) - 1$ do
8: \hspace{1em} $\Delta k \leftarrow r - k_0$
9: \hspace{1em} if $\lambda \cdot \frac{\Delta k}{R(m, q, d_0) - k_0} \geq p_c$ then
10: \hspace{2em} return $q' = (\text{loc}, \text{doc}, \vec{w}, k', d')$
11: for $i = 1$ to $r$ do
12: \hspace{1em} $\alpha' \leftarrow \theta[i]$
13: \hspace{1em} $\beta' \leftarrow \alpha' + ((\theta[R(m, q, d_0) - 1] - (r - i]) - \theta[i] + 2\pi) \% 2\pi)$
14: \hspace{1em} compute the penalty $p$ for the current candidate direction according to Eqn. (5.2.6)
15: \hspace{1em} if $p < p_c$ then
16: \hspace{2em} $k' \leftarrow r$, $d' \leftarrow (\alpha', \beta')$, $p_c \leftarrow p$
17: return $q' \leftarrow (\text{loc}, \text{doc}, \vec{w}, k', d')$

Function CalDirection(q, o)

INPUT: Original query $q$, object $o$

OUTPUT: $\theta_o$ // the direction of $o$ w.r.t. $q$

18: $X \leftarrow o.x - q.x$, $Y \leftarrow o.y - q.y$
19: if $X = 0$ then
20: \hspace{1em} if $Y < 0$ then return $-\pi/2$
21: \hspace{2em} else return $\pi/2$
22: \hspace{1em} $\theta_o \leftarrow \text{Arctan}(Y/X)$
23: if $X < 0$ and $Y \geq 0$ then
24: \hspace{1em} $\theta_o \leftarrow \theta_o + \pi$
25: if $X < 0$ and $Y < 0$ then
26: \hspace{1em} $\theta_o \leftarrow \theta_o - \pi$
27: return $\theta_o$
directions, we sort m’s dominators clockwise according to their angles w.r.t. m’s angle. The very basic refined query is a top-7 query with no query direction. We enumerate each possible refined k’, which is from 3 to 6. For each k’, we find the candidate directions that rank m as top-k’ according to Proposition 5.2.2 and compute their penalties to check whether they are better than the known refined queries. For example, the candidate directions that give m a rank of 3 are: (θ[1], θ[4]), (θ[2], θ[5]), and (θ[3], θ[6]).

We next analyze the time complexity of the proposed algorithm.

**Theorem 5.2.3.** The time complexity of Algorithm 7 is $O(SKT(r) + r^2)$, where $r = R(m, q, d_0)$ denotes the rank of the missing object under the initial query and SKT(r) denotes the time complexity of a spatial keyword top-k query in retrieving the top-r objects.

**Proof.** The algorithm has two phases: (i) compute the initial rank of the missing object; (ii) find the best refined query. The first phase takes advantage of an existing spatial keyword top-k querying algorithm and has time complexity $O(SKT(r))$. For the second phase, we enumerate each possible refined $k’$ from $k_0$ to $r - 1$. For each $k’$, we need to verify $k’$ candidate directions. Thus, in the worst case, the time complexity of this phase is $O(r^2)$. 

**Optimizations:** We remark that optimizations can be applied to both phases in the processing of a why-not query. To speed up the computation of the initial rank of $m$, we buffer the result and internal data structures of the initial query and proceed to process it if a follow-up why-not question is posed. For the second phase, we enumerate each possible refined $k’$ in increasing order so that we can stop the enumeration early if the penalty of the next $k’$ in modifying $k_0$ exceeds that of the currently seen best refined query.
5.3 Answering Why-Not Questions: General Case

We proceed to study how to answer why-not questions on general direction-aware spatial keyword queries. Here, initial queries are specified with search direction requirements. In response to a follow-up why-not question, the system provides the user with a more precise direction so that the refined query can retrieve more useful results.

5.3.1 Case Analysis

Recall that we aim to find the refined query that minimally modifies the initial query and achieves the inclusion of the user’s expected but missing object in its result. Eqn. (5.1.3) quantifies the penalty of a refined query when modifying the direction $d_0 = (\alpha_0, \beta_0)$ and the parameter $k_0$. Unlike the special case in Section 5.2, the bisector of the initial direction is fixed in the general case, i.e., $\frac{\alpha_0 + \beta_0}{2}$. According to Eqn. (5.1.4), the maximum possible modification of the direction is as follows:

$$
\Delta d_{max} = \gamma \cdot \pi + (1 - \gamma) \cdot \max\{\beta_0 - \alpha_0, 2\pi - (\beta_0 - \alpha_0)\}
$$

(5.3.7)

This is obtained when both rotating the initial direction and changing the size of $d_0$ reach the maximum values, i.e., $\pi$ and $\max\{\beta_0 - \alpha_0, 2\pi - (\beta_0 - \alpha_0)\}$, respectively.

The reason why an expected object $m$ is missing from the result of an initial query $q = (loc, doc, \vec{w}, k_0, d_0)$ can be discussed in two scenarios: (i) $m$ is in $q$’s query direction $d_0$, but has a worse rank than $k_0$, i.e., $R(m, q, d_0) > k_0$; (ii) $m$ is outside $q$’s direction $d_0$. Fig. 5.4 illustrates an expected object $m$ missing from a query result in different cases. Assume that the initial query $q$ is a top-2 query and the
points in the figure denote the objects that have better ranking scores than \( m \) w.r.t. \( q \). Obviously, case \( 1 \) belongs to scenario \((i)\), as \( m \) is in the query direction but has a rank of 3. We further divide scenario \((ii)\) into two sub-cases, \( i.e., 2 \) and \( 3 \), according to the “distance” of \( m \)’s angle \( i.e., \theta_m \) to \( \alpha_0 \) and \( \beta_0 \). We measure the “distance” as the length of the direction interval from \( \beta_0 \) to \( \theta_m \) or from \( \theta_m \) to \( \alpha_0 \). In \( 2 \), \( \theta_m \) is closer to \( \beta_0 \), since \((\theta_m - \beta_0 + 2\pi)\%2\pi\) is smaller than \((\alpha_0 - \theta_m + 2\pi)\%2\pi\).

In \( 3 \), \( \theta_m \) is closer to \( \alpha_0 \).

Next, we analyze the problem in these cases.

**Case 1**

Here the missing object is in the query direction. In other words, the missing object \( m \) does have a rank \( R(m, q, d_0) \) under the initial query. Thus, one very basic refined query that can revive \( m \) is to keep the initial direction \( i.e., \Delta d = 0 \) and simply enlarge \( k_0 \) to \( R(m, q, d_0) \). According to Lemma 5.1.3, we must set \( k' = R(m, q, d_0) \) to obtain the smallest penalty, and its corresponding \( \Delta k = R(m, q, d_0) - k_0 \). Any other refined queries with \( \Delta d > 0 \) and \( \Delta k \geq R(m, q, d_0) - k_0 \) have larger penalties and have no chance to be the best refined query. Hence, the maximum possible modification to \( k \) is \( \Delta k_{max} = R(m, q, d_0) - k_0 \). As such, the penalty function for the why-not problem in Case 1 is the following:

\[
\text{Penalty}(q, q') = \lambda \cdot \frac{\Delta k}{R(m, q, d_0) - k_0} + (1 - \lambda) \cdot \frac{\Delta d}{\gamma \cdot \pi + (1 - \gamma) \cdot \max\{\beta_0 - \alpha_0, 2\pi - (\beta_0 - \alpha_0)\}}
\]  

(5.3.8)

Like the special case in Section 5.2, the ranking score of each object remains
unchanged for the refined query. The rank of the missing object \( m \) is determined by the number of objects that dominate \( m \) in the refined direction. Similar to the special case, we can first find the objects that have a ranking score larger than \( m \) in all directions, then determine the best direction for each possible refined \( k' \), and finally return the one with the smallest penalty. Nevertheless, to avoid exploring all query directions around the query location, we prove that the best refined direction belongs to a small search space.

**Theorem 5.3.1.** Consider an initial query \( q \) with direction \( d_0 = (\alpha_0, \beta_0) \) and a missing object \( m \) in \( d_0 \). Let \( \theta_m \) be the angle of \( m \) w.r.t. \( q \). A direction \((\alpha', \beta')\) can result in the best refined query if: (i) \( \theta_m \in (\alpha', \beta') \); and (ii) \((\alpha', \beta') \subset [\theta_m - (\beta_0 - \alpha_0), \theta_m + (\beta_0 - \alpha_0)]\).

**Proof.** The first condition is obviously necessary. It ensures that the missing object \( m \) is in the refined query direction. Proving the second condition is equivalent to proving that (a) \( \alpha' \in [\theta_m - (\beta_0 - \alpha_0), \theta_m] \), and (b) \( \beta' \in [\theta_m, \theta_m + (\beta_0 - \alpha_0)] \). We prove these two by contradiction.

(a) Assume \( \alpha' < \theta_m - (\beta_0 - \alpha_0) \) and \( \alpha' \) is a candidate start angle for the best refined query. Then, the possible refined end angle \( \beta' \) can be considered in two cases: \( \beta' \in [\theta_m, 2\beta_0 - \theta_m] \) or \( \beta' \in (2\beta_0 - \theta_m, \alpha' + 2\pi] \).

In the first case, consider a refined start angle \( \alpha'' = \theta_m - (\beta_0 - \alpha_0) \) and compare the two refined queries with directions \([\alpha', \beta']\) and \([\alpha'', \beta']\). Since both \([\alpha', \beta']\) and \([\alpha'', \beta']\) have sizes no less than the size of the initial direction and \([\alpha'', \beta'] \subset [\alpha', \beta']\), \( \Delta s_{[\alpha'', \beta']} < \Delta s_{[\alpha', \beta']} \). And for rotating the initial direction, \( \Delta r_{[\alpha'', \beta']} - \Delta r_{[\alpha', \beta']} = \left| \frac{\alpha'' + \beta'}{2} - \frac{\alpha_0 + \beta_0}{2} \right| - \left| \frac{\alpha' + \beta'}{2} - \frac{\alpha_0 + \beta_0}{2} \right| = \frac{\alpha' - \alpha''}{2} < 0 \). Hence, \( \Delta r_{[\alpha'', \beta']} < \Delta r_{[\alpha', \beta']} \). Combining these two facts, we can deduce that \( \Delta d_{[\alpha'', \beta']} < \Delta d_{[\alpha', \beta']} \). Moreover, since \([\alpha'', \beta']\) is a subset of \([\alpha', \beta']\), the dominators of \( m \) located in \([\alpha'', \beta']\) must also exist in \([\alpha', \beta']\), which infers that the refined \( k' \) for \([\alpha'', \beta']\) is no larger than that for \([\alpha', \beta']\). Therefore, it follows that \( \Delta k_{[\alpha'', \beta']} \leq \Delta k_{[\alpha', \beta']} \). As both \( \Delta d_{[\alpha'', \beta']} \leq \Delta d_{[\alpha', \beta']} \) and \( \Delta k_{[\alpha'', \beta']} \leq \Delta k_{[\alpha', \beta']} \), the penalty of the refined query with the refined direction \([\alpha'', \beta']\) would be smaller. Thus, \( \alpha' < \theta_m - (\beta_0 - \alpha_0) \) cannot be a candidate start.
angle for an end angle $\beta' \in [\theta_m, 2\beta_0 - \theta_m]$.

In the second case, consider a refined direction with $\alpha'' = \theta_m - (\beta_0 - \alpha_0)$ and $\beta'' = 2\beta_0 - \theta_m$. As both $[\alpha'', \beta'']$ and $[\alpha', \beta']$ are larger than the initial direction and $[\alpha'', \beta''] \subset [\alpha', \beta']$, $\Delta s_{[\alpha'', \beta'']} < \Delta s_{[\alpha', \beta']}$ and $\Delta k_{[\alpha'', \beta'']} \leq \Delta k_{[\alpha', \beta']}$. In addition, $\Delta r_{[\alpha'', \beta'']} = |\theta_m - (\alpha'' + 2\beta_0 - \theta_m - \alpha_0 + \beta_2)| = 0 \leq \Delta r_{[\alpha', \beta']}$. Thus, the refined query with direction $[\alpha'', \beta'']$ would have a smaller penalty. Any direction $[\alpha', \beta']$ with $\alpha' < \theta_m - (\beta_0 - \alpha_0)$ and $\beta' \in (2\beta_0 - \theta_m, \alpha' + 2\pi)$ has no chance to be the best refined direction.

The proof of (a) follows.

The proof of (b) is similar and hence omitted in the interest of space. \qed

Theorem 5.3.1 reduces the search space for the refined directions to a smaller candidate space, making the processing of the why-not query more efficient. We denote this candidate space as $CS(d')$, i.e., $CS(d') = [\theta_m - (\beta_0 - \alpha_0), \theta_m + (\beta_0 - \alpha_0)]$.

Case (2)

Similar to Case (1), the search space for Case (2) can be reduced by Theorem 5.3.2 to a smaller candidate space $CS(d') = [\alpha_0 - (\theta_m - \beta_0), \theta_m + (\beta_0 - \alpha_0)]$.

**Theorem 5.3.2.** Consider an initial query $q$ with a direction $d_0 = (\alpha_0, \beta_0)$ and a missing object $m$ outside $d_0$. Let $\theta_m$ denote the angle of $m$ w.r.t. $q$. If $(\theta_m - \beta_0 + 2\pi) \% 2\pi < (\alpha_0 - \theta_m + 2\pi) \% 2\pi$, a direction $(\alpha', \beta')$ can result in the best refined query if: (i) $\theta_m \in (\alpha', \beta')$; and (ii) $(\alpha', \beta') \subset [\alpha_0 - (\theta_m - \beta_0), \theta_m + (\beta_0 - \alpha_0)]$.

**Proof.** We omit the proof as it is similar to that of Theorem 5.3.1. \qed

Unlike Case (1), in this case the expected but missing object is outside the initial direction. The expected object is filtered out by $d_0$ and does not have a rank under the initial query. It is thus impossible to revive the missing object by simply enlarging $k$ without modifying the initial direction. Nevertheless, Theorem 5.3.2 implies that the refined direction belongs to $CS(d')$. That is, the rank of $m$ can only be influenced by the objects in $CS(d')$ that score better than $m$. The refined
$k'$ can never exceed the rank of the missing object in the whole candidate direction space. Thus, we set the maximum possible refined $k'$ to be the rank of $m$ in $CS(d')$, denoted by $R(m, q, CS(d'))$. As this rank could be smaller than $k_0$, to normalize $\Delta k$ and avoid the denominator $\Delta k_{\text{max}}$ being 0, the maximum modification on $k$ is given as follows:

$$\Delta k_{\text{max}} = \max \{ R(m, q, CS(d')) - k_0, 1 \}$$

Thus, the penalty function for the why-not problem in Case (2) is represented as follows:

$$\text{Penalty}(q, q') = \lambda \cdot \frac{\Delta k}{\max \{ R(m, q, CS(d')) - k_0, 1 \}}$$

$$+ (1 - \lambda) \cdot \frac{\Delta d}{\gamma \cdot \pi + (1 - \gamma) \cdot \max \{ \beta_0 - \alpha_0, 2\pi - (\beta_0 - \alpha_0) \}}.$$ (5.3.9)

Case (3)

Case (3) is symmetric to Case (2) except that the candidate space for the refined direction in this case is $CS(d') = [\theta_m - (\beta_0 - \alpha_0), \beta_0 + (\alpha_0 - \theta_m)]$.

5.3.2 Answering Why-Not

The above discussions analyze the why-not query problem in different cases and provide theorems that reduce the search space. Based on this, we propose an algorithm for solving direction-aware why-not questions in the general case. The pseudo-code is given in Algorithm 8. First, we identify which case the problem falls into according to the relationship between $\theta_m$ and $d_0$ (Line 2). Then the candidate search space $CS(d')$, the maximum possible $\Delta k$, and the initially refined $(d', k')$ for the corresponding case are computed accordingly (Lines 3–11). Next, we enumerate each possible refined $k'$ and compute the refined query with the smallest penalty for each of them (Lines 16–31). Finally, we return the best one as the result (Line 33).

Unlike the special case in Section 5.2, the refined start and end angles do not have to be angles of dominators of the missing object. Instead, a candidate refined direction $d' = [\alpha', \beta']$ that ranks the missing object at a given $k'$ belongs to several range pairs of $\alpha'$ and $\beta'$. See Fig 5.5 as an example. With either pair of $\alpha' \in \{0, 2\pi\}$ and $\beta' \in \{0, 2\pi\}$, this refined query is best to be avoided.
**Algorithm 8 Answering Why-not Questions: General Case**

**INPUT:** Original query \( q = (\text{loc}, \text{doc}, w_i, k_0, d_0) \).

**OUTPUT:** Best refined query \( q' = (\text{loc}, \text{doc}, w_i, k', d') \)

1. \( \theta_m \leftarrow \text{CalDirection}(q, m) \)
2. identify which case the problem falls into according to the relationship between \( \theta_m \) and \( d_0 \)
3. compute the corresponding \( CS(d') = [\alpha_L, \beta_U] \)
4. determine \( R(m, q, CS(d')) \) and record \( m \)'s dominators in set \( S \)
5. if Case 1 then
   6. \( \Delta k_{\text{max}} \leftarrow R(m, q, d_0) - k_0 \)
   7. \( d' \leftarrow d_0, k' \leftarrow R(m, q, d_0), p_c \leftarrow \lambda \)
8. else
9. \( \Delta k_{\text{max}} \leftarrow \max\{R(m, q, CS(d')) - k_0, 1\} \)
10. \( d' \leftarrow CS(d'), k' \leftarrow R(m, q, CS(d')) \)
11. \( p_c \leftarrow \text{penalty}(q, q') \)
12. for each \( \alpha_i \in S \)
13. \( \theta[i] \leftarrow \text{CalDirection}(q, \alpha_i) \)
14. sort \( \theta[i] \) in clockwise order w.r.t. \( \theta_m \)
15. \( \theta[0] \leftarrow \theta_m, \theta[S + 1] \leftarrow \theta_m \)
16. for \( r \) from 1 to \( R(m, q, CS(d')) \)
17. \( \Delta k \leftarrow \max\{r - k_0, 0\} \)
18. if \( \lambda \cdot \Delta k_{\text{max}} \geq p_c \) then
19. \( \text{return } q' = (\text{loc}, \text{doc}, w_i, k', d') \)
20. for \( i \) from 1 to \( r \) do
21. if \( \theta[i-1] \notin [\alpha_L, \theta_m] \) break
22. \( \alpha_L' \leftarrow \theta[i-1], \theta_L' \leftarrow \theta[i-1] + (\theta[S] - (r - i) + 1 - \theta[i-1] + 2\pi)\%2\pi \)
23. if \( \theta[i] \in [\alpha_L, \theta_m] \) then \( \alpha_L' \leftarrow \theta[i] \)
24. else \( \alpha_L' \leftarrow \alpha_L \)
25. if \( \theta[S] - (r - i) \in [\theta_m, \beta_U] \) then \( \beta_U' \leftarrow \theta[i-1] + (\theta[S] - (r - i) - \theta[i-1] + 2\pi)\%2\pi \)
26. else \( \beta_U' \leftarrow \theta[i-1] + (\beta_U - \theta[i-1] + 2\pi)\%2\pi \)
27. solve the linear programming problem with the objective function as \( \Delta d \) and with the constraints as: (i) \( \alpha' \in (\alpha_L', \alpha_U') \); (ii) \( \beta' \in [\beta_L', \beta_U'] \)
28. \( [\alpha', \beta'] \leftarrow \text{the point that minimizes } \Delta d \)
29. compute the penalty \( p \) for the current candidate direction according to the corresponding penalty function
30. if \( p < p_c \) then
31. \( k' \leftarrow r, d' \leftarrow [\alpha', \beta'], p_c \leftarrow p \)
32. \( k' \leftarrow \max\{k', k_0\} \)
33. \( \text{return } q' = (\text{loc}, \text{doc}, w_i, k', d') \)

\((\theta[1], \theta_m], \beta' \in [\theta[4], \theta[3]] \) or \( \alpha' \in (\theta[2], \theta[1]], \beta' \in [\theta_m, \theta[4]] \), the missing object \( m \) has a rank of 2. More generally, let \( CS(d') = [\alpha_L, \beta_U] \), and let \( S \) be the set of \( m \)'s dominators in the candidate search space. We denote the angle of each object \( \alpha_i \) in \( S \) as \( \theta[i] \) and sort them clockwise w.r.t \( \theta_m \). Let \( t \) be the number of objects in \( S \) and
in the direction $[\alpha_L, \theta_m]$. For a given $k'$, the possible $\alpha'$ ranges can be enumerated as $(\theta[i], \theta[i - 1]) \cap [\alpha_L, \theta_m]$, $1 \leq i \leq \min\{k', t + 1\}$, where $\theta[0]$ is set as $\theta_m$. The corresponding $\beta'$ range for an $\alpha'$ range can then be selected accordingly by ensuring that $k' - 1$ dominators of $m$ exist in $[\alpha', \beta']$. There are at most $k'$ such $\alpha'$ and $\beta'$ range pairs for a given $k'$. For example, in Fig. 5.5, there are two such pairs that rank $m$ at 2, as discussed above.

To compute the optimal refined query for a given $k'$, we solve a linear programming problem for each range pair of $\alpha'$ and $\beta'$, with the objective function set as the corresponding penalty function (Line 27). Given an initial query $q$ and a refined $k'$, $\Delta k$ is determined. Minimizing the penalty function is equivalent to minimizing $\Delta d$. Specifically, the linear programming problem of finding the optimal refined direction for a range pair of $\alpha'$ and $\beta'$ that ranks the missing object $m$ at a given rank $k'$ can be modeled as follows:

$$
\begin{align*}
\min \Delta d &= \gamma \cdot \left| \frac{\alpha' + \beta'}{2} - \frac{\alpha_0 + \beta_0}{2} \right| \\
&\quad + (1 - \gamma) \cdot |(\beta' - \alpha') - (\beta_0 - \alpha_0)| \\
\text{s.t.} \quad &\alpha' \in (\theta[i], \theta[i - 1]) \\
&\beta' \in [\theta[i - 1] + (\theta[S] - (k' - i) + 1) - \theta[i - 1] + 2\pi)\%2\pi, \\
&\quad \theta[i - 1] + (\theta[S] - (k' - i)) - \theta[i - 1] + 2\pi)\%2\pi),
\end{align*}
$$

where $i \in \mathbb{Z}$ and $1 \leq i \leq \min\{k', t + 1\}$, and $\alpha_0$ and $\beta_0$ are the start and end angles.

---

3For clarify of the presentation, here we omit the criteria that $\alpha' \in [\alpha_L, \theta_m]$ and $\beta' \in [\theta_m, \beta_U]$. 

---

Figure 5.5: Ranges of $\alpha$ and $\beta$ that rank $m$ at 2
of the initial query’s direction. Note that the $\beta'$ range is computed in a way that ensures that it satisfies the constraint $\beta' \in [\alpha', \alpha' + 2\pi)$.

**Example 11.** Fig. 5.5 shows an example, where two range pairs of $\alpha$ and $\beta$ rank $m$ at $2$. The first pair is $\alpha' \in (\theta[1], \theta_m]$ and $\beta' \in [\theta[4], \theta[3])$ (Fig. 5.5(a)). To ensure that the refined $\beta'$ is in $[\alpha', \alpha' + 2\pi)$, the range of $\beta'$ is measured from $\theta_m$, i.e., $[\theta_m + (\theta[4] - \theta_m + 2\pi)\%2\pi, \theta_m + (\theta[3] - \theta_m + 2\pi)\%2\pi)$. The optimal refined $\alpha'$ and $\beta'$ in this range are then computed by solving the linear programming problem of minimizing $\Delta d$. It is computed similarly for the second range pair of $\alpha'$ and $\beta'$ (Fig. 5.5(b)). By comparing these two results, we then find the best $\alpha'$ and $\beta'$ that rank $m$ at $2$.

We give the time complexity of Algorithm 8 in the next theorem.

**Theorem 5.3.3.** Let $r = R(m, q, CS(d'))$ denote the rank of the missing object in the candidate search space, and let $DSKT(r)$ denote the time complexity of a direction-aware spatial keyword top-$k$ query in retrieving the top-$r$ objects. Algorithm 8 has a time complexity of $O(DSKT(r) + r^2)$.

**Proof.** The proof is similar to that of Algorithm 7. The difference is that in Algorithm 8, we enumerate the possible $k'$ from $1$ to $r$, and for each $k'$, there exist at most $k'$ range pairs of refined $\alpha'$ and $\beta'$. We need to solve a linear programming problem for each range pair. Nevertheless, the time complexity of solving such a linear programming problem is constant, as the optimal point is always on the vertices of the convex polygon. Thus, the time complexity of computing the refined query is also $O(r^2)$.

**Optimizations:** The optimizations proposed in Section 5.2 are also applicable to Algorithm 8.

### 5.4 Handling Multiple Missing Objects

Next, we extend the proposed algorithms to support why-not queries with a set $M$ of missing objects. Recall that we refine the query direction $d$ and the result
cardinality $k$ to revive the missing objects. As the query direction works as a filter, only the objects in the direction are considered as candidates for a query result. To achieve the inclusion of all missing objects, it is a basic requirement that the refined direction $d'$ covers all missing objects, i.e., $\forall m_i \in M (\theta_m \in d')$.

Another requirement is that, the refined $k'$ should be able to get the inclusion of the missing object with the worst rank in the refined $d'$, i.e., $k' \geq R(M, q, d')$, where $R(M, q, d') = \max_{m_i \in M} R(m_i, q, d')$.

With these observations, the already proposed algorithms are extended to support multiple missing objects as follows: (i) find the sufficient directions $d_s$'s that cover all the missing objects; (ii) for each sufficient direction $d_s$, enumerate the possible refined $k'$ to find the optimal refined query; (iii) determine the best refined query by comparing the optimal refined queries for all sufficient directions. Let $\theta_m[i]$ denote the angle of a missing object $m_i$ in $M$ and sort the objects in increasing order of their angles. Assume $\theta_m[0] = \theta_m[|M|]$. A sufficient direction that covers all the missing objects in $M$ is defined as $[\theta_m[i], \theta_m[i-1]]$, where $1 \leq i \leq |M|$. See Fig. 5.6(a) for an example. Here, the missing object set is $M = \{m_1, m_2\}$, and the dominators of the worst ranked object in $M$ are marked as black points. Next, $d_{s_1} = [\theta_m[1], \theta_m[2]]$ and $d_{s_2} = [\theta_m[2], \theta_m[1]]$ are two possible sufficient directions for $M$. Consider $d_{s_1} = [\theta_m[1], \theta_m[2]]$. To find the best refined query, we enumerate the possible refined $k'$ starting from $R(M, q, d_{s_1})$, i.e., 3. The process of finding the optimal refined direction for a refined $k'$ is the same as that for a single missing object, i.e., comparing the directions with start and end angles as the angles of the dominators in the special case and solving linear programming problems in the general case. We need to find the optimal refined direction for each sufficient direction of $M$, as users' preferences of modifying $k$ vs. $d$ and the distributions of the dominators of the worst ranked missing object vary. See Fig. 5.6(b) for an example. Assume a top-2 query is initially issued. If we only consider the sufficient direction $[\theta_m[1], \theta_m[2]]$, the smallest refined $k'$ equals $R(M, q, [\theta_m[1], \theta_m[2]]) = 6$. In contrast, the smallest refined $k'$ for another sufficient direction $[\theta_m[2], \theta_m[1]]$ is only 2, which
incurs a smaller penalty for modifying $k$.

**Theorem 5.4.1.** Let $r = R(M, q, [-\pi, \pi])$ denote the lowest rank of the missing objects in the whole database and $SKT(r)$ denote the time complexity of a spatial keyword top-$k$ query retrieving the top-$r$ objects. The time complexity of the why-not algorithm for a set $M$ of missing objects is $O(SKT(r) + |M| \cdot r^2)$.

**Proof.** As we need to consider all sufficient directions, a traditional spatial keyword top-$k$ query needs to be processed to compute the dominators of the objects in $M$ in all directions. Moreover, an optimal refined query is sought for each sufficient direction. If no two missing objects are in the same direction, there are $|M|$ sufficient directions for a missing object set $M$. Thus, the overall time complexity is $O(SKT(r) + |M| \cdot r^2)$. \qed

**Optimizations:** In addition to the optimizations that we proposed for a single missing object, we enumerate the sufficient directions in increasing order of their ranks of the worst ranked object in $M$, i.e., $R(M, q, d_s)$, so that we can early stop the enumeration if the smallest refined $k'$ for the next $d_s$, i.e., $R(M, q, d_s)$, has a larger penalty in modifying $k$ than the penalty of the currently seen best refined query.

## 5.5 Empirical Study

This section evaluates the effectiveness and efficiency of the proposed algorithms.
### 5.5.1 Experimental Setup

**System Setup and Metrics**

Our experiments are all conducted on a PC with an Intel Core i5 2.7GHz CPU and 8GB memory running Windows 7 OS. The algorithms are implemented in Java, and the maximum main memory of the Java Virtual Machine is set to 4GB. The proposed why-not query techniques are applicable to any direction-aware spatial keyword top-

$k$ query algorithm. In our experiments, we extend an existing algorithm [14] to process direction-aware spatial keyword top-$k$ queries using the language model by examining whether each accessed MBR or object is in the query direction. The index structure adopted, *i.e.*, the IR-tree, is disk-resident. The page size is set to 4KB, the buffer size is set to 4MB, and the capacity of a node is set to 100. Note that the proposed algorithms for processing why-not questions are independent of the algorithm for the direction-aware spatial keyword search. For each set of experiments, we randomly generate 1,000 queries and report the average query time.

**Datasets**

We study the performance of the proposed algorithms using two real datasets, EURO and GN. Both are used widely in spatial keyword related research [5, 29]. Each dataset contains a number of objects with a spatial location and a set of keywords. EURO is a dataset of points of interest such as ATMs, hotels, and stores in Europe (www.allstays.com); and GN is obtained from the US Board on Geographic Names (geonames.usgs.gov) and contains a set of geographic objects. Table 5.1 gives more details about the datasets.

**Parameters**

We evaluate the performance of our algorithms when varying different parameters. Table 5.2 lists these parameters, where the default values are highlighted in bold. By default, the why-not question is issued for a missing object that ranks at $10 \cdot k_0 + 1$ under the initial query without taking into account a query direction. As such, the
missing object might be located inside or outside the user-specified query direction. As a default, we fix the weighting factor $\gamma$ in Eqn. (5.1.4) to 0.5 to balance the changes between the rotation and the size of a refined direction. For each set of experiments, the parameters are set to their default values unless specified otherwise.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_0$</td>
<td>1, 3, 10, 30, 100</td>
</tr>
<tr>
<td># of keywords</td>
<td>2, 4, 6, 8, 16</td>
</tr>
<tr>
<td>$\bar{w}$</td>
<td>$&lt;0.1, 0.9&gt;, &lt;0.3, 0.7&gt;$, $&lt;0.5, 0.5&gt;, &lt;0.7, 0.3&gt;, &lt;0.9, 0.1&gt;$</td>
</tr>
<tr>
<td>$R(m, q, [-\pi, \pi])$</td>
<td>11, 31, 101, 301, 1001</td>
</tr>
<tr>
<td>size of $d_0$</td>
<td>$\frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}, \pi, 2\pi$</td>
</tr>
<tr>
<td># of missing objects</td>
<td>1, 3, 10, 30</td>
</tr>
</tbody>
</table>
top-10 query with no query direction. The candidate directions for the baseline algorithm are obtained according to Proposition 5.2.2. Fig. 5.7 shows the runtime of the algorithms under different penalty options, where $PMK$ stands for “Prefer Modifying K ($\lambda = 0.1$)”; $PMD$ stands for “Prefer Modifying Direction ($\lambda = 0.9$)”; and $NM$ stands for “Never Mind ($\lambda = 0.5$)”. As we can see, Algorithm 7 outperforms the baseline method by two orders of magnitude. That is mainly because the baseline method needs to invoke a spatial keyword top-$k$ query to determine the rank of the missing object for each candidate direction, which incurs high computation cost. In contrast, Algorithm 7 is able to calculate the penalty of a candidate direction in constant time.

![Figure 5.7: Running time vs. Baseline & Algorithm 7](image)

In the remaining experiments, we examine the performance of the algorithm (i.e., Algorithm 2) proposed for the general case. We do not include the baseline algorithm for comparison since, as explained above, it does not work for the general case.

**Varying $k_0$**

In this set of experiments, we investigate how different values of the parameter $k_0$ under the initial query affects the performance of the algorithm. In our setting, the rank of the missing object under the initial query varies with $k_0$, i.e., $R(m, q, [-\pi, \pi]) = 10 \cdot k_0 + 1$. For instance, when an initial top-3 query is issued, the corresponding why-not question seeks to revive the object ranked at 31 in the
database \textit{w.r.t.} the initial query. Fig. 5.8 plots the results. Recall that the proposed algorithm consists of two phases, \textit{i.e.}, \textit{i)} computing the initial rank of the missing object and \textit{ii)} finding the best refined query. Both of the two phases take more time when the missing object has a worse rank. In our setting, the rank of the missing object gets worse when $k_0$ increases; hence, the runtime of the algorithm increases with $k_0$. However, the algorithm scales well with the increase of $k_0$. For instance, when $k_0 = 10$, the missing object ranks at 1001; the algorithm is able to process the why-not query within 400ms and 1.8s on datasets Euro and GN, respectively.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.8.png}
\caption{Running time vs. $k_0$}
\end{figure}

\section*{Varying the number of query keywords}

We next evaluate the effect of different numbers of query keywords. Fig. 5.9 plots the results. Intuitively, the number of query keywords only affects the first phase of the algorithm, \textit{i.e.}, computing the missing object’s rank in the candidate search space. As having more query keyword requires more time to compute the textual similarities between the query keywords and index tree nodes/objects, the runtime shows an increasing tendency when the number of query keywords increases.

\section*{Varying $\vec{w}$}

The weighting vector $\vec{w}$ allows users to set their preferences between spatial proximity and textual relevance when issuing a spatial keyword top-$k$ query. We evaluate the performance of the proposed algorithm by varying $\vec{w}$ in this set of experiments.
Different settings of $\bar{w}$ also only affect the algorithm in the first phase. As we can see from Fig. 5.10, the query time decreases when $w_s$ in $\bar{w}$ is increased. The reason is that, a smaller $w_s$ means a higher weight to textual relevance, which lowers the importance of spatial proximity in the ranking function. Consequently, the pruning ability of the IR-tree decreases and more tree nodes need to be accessed.

Varying $d_0$

Next, we investigate the performance of the algorithm when varying the initial query direction $d_0$. In the experiments, queries with different initial direction sizes, from $\frac{\pi}{6}$ to $2\pi$, are generated randomly. The average query time is shown in Fig. 5.11. A larger $d_0$ results in a larger candidate space $CS(d')$ for the refined query directions. As the processing of a direction-aware spatial keyword query consumes more time for a larger search space, the time for computing the rank of the missing object in a
larger candidate space increases. This explains why the runtime increases with the size of $d_0$. Nevertheless, the algorithm scales well as $d_0$ increases. For example, the running time only increases 20% when the direction varies from $\frac{\pi}{6}$ to $2\pi$.

![Figure 5.11: Running time vs. $d_0$](image)

**Varying the initial rank of the missing object**

We also study the performance of our algorithm when issuing why-not questions for missing objects with different ranks in the initial query. In the experiments, a default top-10 query is used. We ask five why-not questions with the missing object being the one ranked at 11, 31, 101, 301, and 1001, respectively. Fig. 5.12 shows the results. As expected, the spatial keyword top-$k$ query consumes more time to compute the rank of the missing object that ranks worse under the initial query. Moreover, a worse ranked missing object results in many more dominators and thus produces more candidate directions, which makes the phase of finding the best refined query take longer as well. These are the reasons for that the runtime increases when the missing object’s rank gets worse.

It is interesting to observe that the result of this set of experiments is quite similar to that of varying $k_0$. This suggests that the initial rank of the missing object affects the performance of the algorithm significantly while $k_0$ has little effect.
Varying the number of missing objects

We next consider the performance of the algorithm when changing the number of missing objects. The initial query is a top-10 query with a direction of size $\frac{\pi}{3}$. Missing objects are selected randomly among the objects ranked between 11 and 101 in the whole database w.r.t. the initial query. Fig. 5.13 shows that the query time increases with more missing objects. The reason is that having more missing objects produces a larger candidate search space for the refined directions. Nevertheless, the increase of the query time is only moderate. This is because only the missing object with the worst initial rank has an impact on the performance.

Scalability

In the last set of experiments, we study the scalability of the proposed algorithm. To do so, we randomly select different numbers of objects from the datasets to test
the performance of the algorithm with different dataset sizes. All parameters for the queries are set to the default values. Fig. 5.14 plots the results. In our setting, the initial rank of the missing object does not change when the dataset size increases, which means that the cost of the phase that finds the best refined query is unaffected. However, the cost of processing a spatial keyword top-$k$ query increases with the dataset size. This is the reason why the query time of the proposed algorithm is sublinear to the increase of dataset size.

Impact of Penalty Options: From all of the above experimental results, we may observe that the user’s penalty preference (i.e., PMK, NM, or PMD) has very small impact on the performance of the proposed algorithms under the considered parameter settings. This is because the penalty preference $\lambda$ only affects the second phase of the algorithms. Since we enumerate the possible refined $k'$ settings in an increasing order, a larger $\lambda$ can help stop the enumeration earlier. This explains the overall trend of the query time with different penalty option settings: $PMK > NM > PMD$.

5.6 Summary

In this chapter, we study the problem of answering why-not questions in the context of spatial keyword top-$k$ queries by refining users’ query directions. We tackle the problem in two cases. Based on insightful problem analysis, we prove that the solution space is a finite set of candidates for the special case with no initial query direction, and we provide linear programming solution for the general case. We also
extend the proposed algorithms to support multiple missing objects. Extensive ex-
eriments with real datasets demonstrate that the proposed algorithms are capable
of superior performance compared to a baseline method and are robust in a broad
range of settings.
Chapter 6

Conclusions and Future Work

The widespread diffusion of smartphones gives prominence to spatial keyword query services. While spatial keyword queries have been extensively studied, state-of-the-art techniques do not provide systematic functionality that allows users to ask why some known object is unexpectedly missing from a query result. Such functionality of answering why-not questions improves the usability of the systems. It provides users an interactive and cooperative experience with the querying systems, where users can question the result of a query and have their queries debugged and fixed automatically. In this dissertation, we formulate the why-not spatial keyword top-$k$ queries from different perspectives towards different application scenarios, and we present our explanation models and efficient algorithms for answering these why-not questions.

6.1 Contributions

The first contribution of this dissertation is that we take the first step to study the why-not spatial keyword top-$k$ queries. We formulate the why-not questions on spatial keyword top-$k$ queries from different perspectives and model them as query refinement problems. We consider the refinement of several parameters that are in some cases difficult for users to input, including the preference weighting vector, the query keywords and the query direction. Each of them has obvious applications
under different scenarios and they together make spatial keyword querying systems more user-friendly.

The second contribution of this dissertation is the efficient algorithms that we proposed for answering these why-not questions. Specifically, for the preference-adjusted why-not questions, we reduce the ranking updates to a geometric problem, reduce the search space to a finite set of candidate vectors and propose an index-based ranking estimation algorithm that prunes candidate weighting vectors and improves query processing performance; for the keyword-adapted why-not questions, we propose both a basic algorithm with a set of optimizations and an index-based bound-and-prune algorithm that are applicable for different textual similarity models; for the direction-aware why-not questions, we provide detailed problem analysis to reduce the solution space and reduce the search of the best refined query to a linear programming problem in general cases. We also extend all these algorithms to support why-not questions with multiple missing objects.

The third contribution of this dissertation is that we conduct extensive experiments on real-life datasets to evaluate the efficiency and effectiveness of our proposed algorithms. The results demonstrate that the algorithms perform well in a broad range of settings. In particular, the results show that our proposed algorithm for the preference-adjusted why-not questions is up to 3 times faster than the baseline algorithm in terms of running time and reduces the I/O cost by up to an order of magnitude; and the algorithm that we proposed for the direction-aware why-not questions is two orders of magnitude faster than a baseline method.

### 6.2 Possible Directions for Future Work

Several interesting directions for future work are worth investigation.

- Firstly, it is relevant to investigate the refinement of the query location to make this line of work more complete. The refinement of the query location has many applications in market analysis and decision support. For example,
by providing more accurate query locations, users can have their target objects ranked higher and thus earn more potential customers. As the potential query location lies in infinite space, new algorithms based on safe zones or similar concepts need to be designed.

- Secondly, we consider the Euclidean space so far. However, the distance between the query location and objects is constrained by a road network in many real applications, which cannot be computed efficiently using the R-tree. It is thus also relevant to consider the road networks in the query, where novel index structures need to be designed.

- Thirdly, the current techniques in this dissertation can only support the refinement of one single parameter, i.e., the preference weighting vector, the query keywords or the query direction. There can be cases that users may want to refine these query parameters simultaneously. It is thus of interest to investigate the why-not problem where refinement on several query parameters is demanded.

- Lastly, other than the why-not questions, users may pose up a why question after receiving a query result, asking why particular data items appear in the result. Providing the functionality of answering such why questions can make the system more usable and interactive. The design of the explanation models and algorithms for the why questions could be an interesting direction for future work.

Based on these, we also plan to build an integrated framework that supports the answering of why-not/why questions on spatial keyword top-k queries while considering different parameters, including the refinement of the preference weighting vector, the query keyword set, the query direction and the query location in a concerted fashion.
Bibliography


CURRICULUM VITAE

Academic qualification of the thesis author, Mr. CHEN Lei:

- Received the degree of Bachelor of Engineering from South China University of Technology, 2012.

December 2016