2018

Towards why-not spatial keyword top-k queries: A direction-aware approach

Lei Chen
Hong Kong Baptist University, psuanle@gmail.com

Yafei Li
Zhengzhou University, yafeics@outlook.com

Jianliang Xu
Hong Kong Baptist University, xujl@hkbu.edu.hk

Christian S. Jensen
Aalborg University, csj@cs.aau.dk

Towards Why-Not Spatial Keyword Top-k Queries: A Direction-Aware Approach

Lei Chen, Yafei Li, Jianliang Xu, *Senior Member, IEEE,* and Christian S. Jensen, *Fellow, IEEE*

Abstract—With the continued proliferation of location-based services, a growing number of web-accessible data objects are geo-tagged and have text descriptions. An important query over such web objects is the *direction-aware spatial keyword query* that 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. In some cases, it can be difficult for users to specify appropriate query parameters. After getting a query result, users may find some desired objects are unexpectedly missing and may therefore question the entire result. Enabling why-not questions in this setting may aid users to retrieve better results, thus improving the overall utility of the query functionality. This paper studies the direction-aware why-not spatial keyword top-k query problem. We propose efficient query refinement techniques to revive missing objects by minimally modifying users' direction-aware queries. We prove that the best refined query directions lie in a finite solution space for a special case and reduce the search for the optimal refinement to a linear programming problem for the general case. Extensive experimental studies demonstrate that the proposed techniques outperform a baseline method by two orders of magnitude and are robust in a broad range of settings.

Index Terms—Why-not questions, spatial keyword top-k queries, query refinement.

1 Introduction

Web objects are becoming increasingly content-rich. In particular, the continued proliferation of location-based services (LBS) and the increasing number of web objects with both textual keywords and spatial location information combine to give prominence to spatial keyword queries [3], [7], [15]. Among them, considering the direction-aware search requirements from many LBS users, a *direction-aware spatial keyword top-k query* [10], [25], [26] takes a user location, a set of keywords, and a search direction as arguments and retrieves the k objects in the search direction that are ranked highest according to a ranking function that considers both spatial proximity and textual similarity. It is relevant to take into account the query direction in a number of scenarios. For example, a user walking to a supermarket may want to find an ATM in his/her walking direction, or a user on a high-way may want to find a gas station or restaurant in his/her general travel direction (i.e., the right front region in right-driving countries); in role-playing games with a first-person perspective, players may want to search the battlefield information such as weapon stores and medical stations in the angle of view.

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. After a user issues a (direction-aware) spatial keyword top-k query and receives the result, the user may find the query result is not as expected and that some desirable objects are unexpectedly missing from the result. This may lead the user to question the overall result and to wonder whether other unknown objects that are relevant to the query are also missing. To enhance the utility of the query functionality, it is relevant to provide explanations about desired but missing objects and to automatically suggest a refined query that includes the desired objects in its result. The motivation and significance of this functionality is illustrated by two examples.

Example 1. *After a busy day of sightseeing, Bob wants to have dinner nearby before walking back to the hotel, which is on the downhill side. He issues a query to find the top-3 nearby "Sushi" restaurants. Surprisingly, he finds that the result contains only restaurants that are on the uphill side; and a restaurant on the downhill way to the hotel that he visited yesterday is not in the result. Bob 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 his way to the hotel? How can he add a search direction so that the missing restaurant and possibly other good options appear in the result?*

Example 2. *In preparation for attending an overseas conference, Clair issues a query to find the top-3 hotels that are near the conference location and are in the direction of the old town. She is surprised that the result contains only local hotels that are unknown to her and that a well-known international hotel in the vicinity is not in the result. Clair wonders whether the exclusion occurs because the search direction is not set properly and how the query direction can be modified so that the expected hotel, as well as potentially other good hotels, appear in the result?*

- *L. Chen, Y. Li and J. Xu are with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong. L. Chen is also with Huawei Noah's Ark Lab, Hong Kong, and Y. Li is also with the School of Information Engineering, Zhengzhou University, Zhengzhou, China.*
  E-mail: {lchen, yafeili, xujl}@comp.hkbu.edu.hk.
- *C. S. Jensen is with the Department of Computer Science, Aalborg University, Denmark.*
  E-mail: csj@cs.aau.dk.

A common pattern in the above scenarios is that the user wants to know *why* an expected object does *not* appear in a result. This type of functionality relates to the query quality and is called *why-not* functionality [6]. Three typical solution models exist: (1) *manipulation identification* [6], which identifies query operators that prevent missing objects from being included in a result; (2) *database modification* [22], [23], which updates the original database so that the query can revive missing objects; and (3) *query refinement* [9], [11], [21], [36], which revises the original query so that missing objects can enter the result. We adopt the query refinement model to answer why-not questions on direction-aware spatial keyword top-k queries. The previous works adjust the preferences between spatial proximity and textual relevance [9] or suggest more accurate query keywords [11] to get the inclusion of missing objects in the query result. In this paper, we address the problem from a new perspective, *i.e.*, modifying the query direction as motivated by the above examples.

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.

The main contributions of this paper are summarized as follows:

- We identify a novel direction-aware why-not spatial keyword top-k query and formulate it as a query refinement problem. To the best of our knowledge, we are the first to study this problem.
- We provide a detailed problem analysis and propose efficient query refinement algorithms to answer direction-aware why-not questions by reducing solution spaces for both the special case and the general case.
- We extend the proposed algorithms to support why-not questions with multiple missing objects.
- We perform extensive experiments on real-life datasets to evaluate the performance of the proposed algorithms. The results indicate that the algorithms are efficient in a broad range of settings. In particular, the proposed solution is two orders of magnitude faster than a baseline method.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces preliminaries and defines the direction-aware why-not spatial keyword query problem. Sections 4 and 5 present the problem analysis and solutions in the two different cases. We extend the algorithms to support queries with multiple missing objects in Section 6. The experimental studies are covered in Section 7. Finally, we conclude in Section 8.

2 Related Work

To the best of our knowledge, no prior studies consider why-not questions on spatial keyword top-k queries from the perspective of query direction refinement. In the following, we survey studies on spatial keyword queries and why-not queries separately, and we relate the studies to the setting of this paper.

2.1 Spatial Keyword Query Processing

A spatial keyword query retrieves the most relevant spatial web objects with respect to both spatial proximity and textual relevance. A number of indexing and query processing techniques have been proposed for this query. The IR-tree [15], [27], [37] is a widely used index structure that integrates R-trees and inverted files. To quickly prune the search space, it supports simultaneous estimation of spatial distance and textual similarity during index access. The IR^2-tree [16] is another hybrid index that combines R-trees with superimposed text signatures. This index is applicable when the keywords serve as a Boolean filter. Rocha-Junior and Nørvåg [34] study the spatial keyword query on road networks. To rank the objects considering both network distance and textual similarity, inverted files are employed to index the documents of the web objects lying on a segment of the road network. A comprehensive comparison of existing spatial keyword indexing techniques is available [7].

Different variants of the spatial keyword query have been studied. Chen *et al*. [13] consider a query that retrieves the web objects which contain the query keywords and whose page footprints intersect with a query footprint. Zhang *et al*. [40], [41] investigate an m-closest keywords (mCK) query to retrieve the spatially closest objects that match user-specified keywords. To efficiently evaluate this query, the bR*-tree and virtual bR*-tree are proposed to augment each node with a bitmap and a set of MBRs for the keywords. Cao *et al*. [4] introduce a query that finds 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 keywords and that have the lowest inter-object distances. Further, Fan *et al.* [18] study the spatial keyword similarity search in regions of interest. Li *et al.* [28] explore a spatial approximate string query that is a range query augmented with a string similarity predicate. Bouros *et al.* [2] identify the 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 answer quality. More recently, Lee *et al.* [31] study the processing and optimizations for main memory spatial keyword queries. Choudhury *et al.* [14] 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.* [35] study location-based keyword search on RDF data. Lin *et al.* [29] and Xie *et al.* [39] aim to find the query keyword sets and the query locations, respectively, to make a target object in the result of a spatial keyword top-k query. Direction-aware spatial keyword queries have been investigated [25], [26], the aim being to find the spatially

closest objects in a given query direction that cover all query keywords. However, none of the above studies address the why-not spatial keyword query problem.

2.2 Why-Not Query Processing

To improve the usability of database systems, the concept of explaining a null answer to a database query was presented by Motro [32], [33], and the *why-not* problem was first introduced by Chapman and Jagadish [6]. Existing approaches towards answering why-not questions can be classified into three main categories. Chapman and Jagadish [6] use *manipulation identification* to identify query operators that eliminate users' desired but missing objects on Select-Project-Join (SPJ) queries. Other studies [22], [23] study the why-not questions on SPJ queries and SPJUA (SPJ + Union + Aggregation) queries by adopting a *database modification* approach, which updates the original database so that the missing objects become part of query results. Tran and Chan [36] propose to retrieve missing objects through *query refinement*, which aims to prompt users how to revise their query parameters to revive their expected objects in the query results. He and Lo [21] employ the query refinement to answer why-not questions on top-k preference queries. They aim to make missing objects enter the result by minimally modifying the original query, where a penalty function measuring the amount of modifications is adopted. More recently, the query refinement model has been applied to answer why-not questions on different queries and data settings, including social image search [1], reverse skyline queries [24], reverse top-k queries [19], and metric probabilistic range queries [8].

In previous work [9], [11], we study the why-not spatial keyword queries from different perspectives. In one study [9], we propose techniques that help users adjust the preferences between the spatial distance and textual similarity; in another [11], we provide users with more precise query keywords. These two why-not functionalities have been integrated into a spatial keyword query system [12]. However, these studies do not consider query direction refinement and are unable to suggest more accurate query directions that better capture users' query intent, which is the focus of this paper.

3 Preliminaries and Problem Formulation

In this section, we formalize the problem of direction-aware why-not spatial keyword top-k queries.

3.1 Spatial Keyword Top-k Queries

Let $\mathcal{D}$ denote a database of spatial objects. Each object $o \in \mathcal{D}$ is associated with a pair (loc, 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 $q.\vec{w} = \langle w_s, w_t \rangle$, 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 $\mathcal{D}$ 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 [15]:

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

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 $\mathcal{D}$. The textual similarity $TSim(o, q)$ can be computed using an information retrieval model [17], 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 [9], [15] in this paper. The larger score computed by Eqn. 1 denotes the higher relevance an object to the query.

Given a query q, the rank of an object o is given in terms of Eqn. (1) as follows:

$$R(o, q) = |\{o' \in \mathcal{D} \mid ST(o', q) > ST(o, q)\}| + 1 \quad (2)$$

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

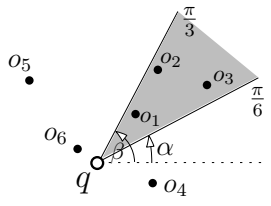
Definition 1. **Spatial Keyword Top-k Query.** A spatial keyword top-k query q returns a set $\mathcal{R}$ of k objects from $\mathcal{D}$, where $\forall o \in \mathcal{R}\ (\forall o' \in \mathcal{D} - \mathcal{R}\ (ST(o, q) \geq ST(o', q)))$, or in terms of object ranking, $\forall o \in \mathcal{R}\ (\forall o' \in \mathcal{D} - \mathcal{R}\ (R(o, q) \leq R(o', q)))$.

While the spatial keyword top-k query has been extensively studied, *direction-aware* search, which is demanded in many LBS scenarios, has recently received more attention [20], [25], [26]. 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. 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 (α, β), denoting that the query is interested only in objects in the direction (α, β).[1] 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 $(loc, doc, \vec{w}, k, d)$.

Definition 2. **Direction-Aware Spatial Keyword Top-k Query.** Let $\mathcal{D}_d$ denote the objects in $\mathcal{D}$ 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 $\mathcal{D}_d$, where $\forall o \in \mathcal{R}\ (\forall o' \in \mathcal{D}_d - \mathcal{R}\ (R(o, q, d) \leq R(o', q, d)))$.

In other words, instead of considering the whole database $\mathcal{D}$, a direction-aware spatial keyword query considers only the objects in a directional range as candidates

1. A direction (α, β) is the angular space passed through by rotation from α to β 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 $\exists n \in \mathbb{Z}, \theta + 2n\pi \in (\alpha, \beta)$.



(a)

	$ST(o,q)$
$\mathbf{o_1}$	0.72
o_2	0.3
$\mathbf{o_3}$	0.58
o_4	0.65
o_5	0.43
o_6	0.83

(b)

Fig. 1. Top k and Direction-Aware Top-k Spatial Keyword Query

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 3. *Fig. 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.*

3.2 Direction-Aware Why-Not Spatial Keyword Query

When issuing a direction-aware spatial keyword query $q = \{loc, doc, \vec{w}, k_0, d_0\}$, 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 $q' = \{loc, doc, \vec{w}, k', d'\}$ 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 [9], [11], [19], [21], [24]. 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 [11], [19], [21] that quantifies the modification as a weighted sum of the changes of parameters Δk and Δd. The penalty of a refined query q' against the initial query q is defined as follows:

$$Penalty(q, q') = \lambda \cdot \frac{\Delta k}{\Delta k_{max}} + (1 - \lambda) \cdot \frac{\Delta d}{\Delta d_{max}}, \quad (3)$$

where $\lambda \in [0, 1]$ is a user preference on modifying k versus d. Here, Δk_{max} and Δd_{max} denote the maximum possible modifications of k and d, respectively. They are used to normalize Δk and Δd 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 4 and 5. 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 α and the end angle β, 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.*, Δr and Δs. Formally, Δd is defined as follows:

$$\begin{aligned} \Delta d &= \gamma \cdot \Delta r + (1 - \gamma) \cdot \Delta s \\ &= \gamma \cdot |\frac{\alpha' + \beta'}{2} - \frac{\alpha_0 + \beta_0}{2}| \\ &\quad + (1 - \gamma) \cdot |(\beta' - \alpha') - (\beta_0 - \alpha_0)| \end{aligned} \quad (4)$$

The rotation of the direction Δr 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 3. **Direction-Aware Why-Not Spatial Keyword Top-k Query.** Given a set $\mathcal{D}$ of spatial objects, a missing object set M, an original direction-aware spatial keyword query $q = (loc, doc, \vec{w}, k_0, d_0)$, the *direction-aware why-not spatial keyword top-k query* returns the refined query $q' = (loc, doc, \vec{w}, k', d')$, with the lowest penalty according to Eqn. (3) and the result of which includes all objects in M.

3.3 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 1 are candidates for the best refined query.

Lemma 1. 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 θ_{m_i} denotes m_i's angle; and *(ii)* $k' = R(M, q, d')$; or $R(M, q, d') \leq k' \leq k_0$, where $R(M, q, d') = max_{m_i \in M} R(m_i, q, d')$.

Proof. The proof is straightforward and hence omitted.

According to Lemma 1, 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 [19], [21]. 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 4 and 5, and we extend the algorithms to support *multiple missing objects* in Section 6.

4 Answering Why-Not Questions: A 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).

4.1 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 (α', β'), we can always find an α_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 Δd can be rewritten as follows:

$$\Delta d = 2\pi - (\beta' - \alpha') \tag{5}$$

Thus, the maximum possible modification of the direction is $\Delta d_{max} = 2\pi$, which is obtained when $\alpha' = \beta'$. Moreover, Δk_{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:

$$Penalty(q, q') = \lambda \cdot \frac{\Delta k}{R(m, q, d_0) - k_0} + (1 - \lambda) \cdot \frac{2\pi - (\beta' - \alpha')}{2\pi} \tag{6}$$

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

4.2 Ranking Updates and Direction Modifications

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 α and an end angle β. To determine a refined direction, we need to identify its start and end angles. Given a start angle α' of a refined direction, the following theorem holds.

Theorem 1. Consider an initial query q, a missing object m, and a refined start angle α'. Let θ_m denote the angle of m w.r.t. q. If a direction (α', β') 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 β' such that $ST(o, q) > ST(m, q)$.

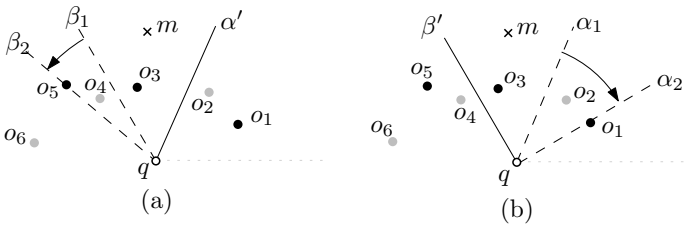


Fig. 2. A candidate end (start) direction for a given α' (β')

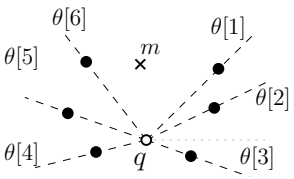
Proof. The first condition is obviously necessary. It ensures that the missing object is in the query direction. We prove the second condition by contradiction. Assume that β' is a candidate end angle and that no object with angle β' scores higher than the missing object m under the initial query q. 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 β'' 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 (α', β') and (α', β''). Thus, Δk is identical for the end angles β' and β''. However, as $(\alpha', \beta') \subset (\alpha', \beta'')$, $\Delta d_{\beta'} > \Delta d_{\beta''}$. According to Eqn. (6), for a given k', a larger direction is preferred. The penalty for the end angle β' exceeds that of β''. Thus, β' cannot be a candidate end angle for start angle α', which contradicts our assumption. Similarly, in the latter case, any angle in $(\beta', \alpha' + 2\pi]$ would have a smaller penalty than β' so that β' cannot be a candidate end angle. Thus, Theorem 1 holds.

Similarly, for a given refined end angle β', the refined start angle α' 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 4. *In Fig. 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 in grey. In Fig. 2(a), for start angle α', when increasing the end angle β' from β_1 to β_2, the rank of m in the direction (α', β') remains unchanged. But as the size of the direction increases, according to Eqn. (6), the penalty for the refined direction (α', β') keeps decreasing. Consequently, no end angle between β_1 and β_2 can yield the best refined direction. The case is similar for the candidate start angles when a refined end angle β' is given (See Fig. 2(b)).*

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

Proposition 1. A refined direction (α', β') can result in the best refined query if: *(i)* $\theta_m \in (\alpha', \beta')$; and *(ii)* both the start angle α' and the end angle β' have an object that



Candidate directions for $k' = 3$:
1: $(\theta[1], \theta[4])$
2: $(\theta[2], \theta[5])$
3: $(\theta[3], \theta[6])$

Fig. 3. Candidate directions for a given refined k'

ranks higher than the missing object under the initial query.

Proposition 1 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 (α', β') 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.

4.3 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 1.

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).[2]

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. These candidate directions are enumerated in ascending order of the number of m's dominators located to the right of m (Line 11). For an enumerated number i, we set the start angle α' and the end angle β' as the extreme cases of $\theta[i+1]$ and $\theta[(R(m, q, d_0) - 1) - (r - i - 1)]$, respectively, to obtain its smallest penalty (Lines 12–13). The end angle β' is further computed to satisfy the constraint that $\alpha' \leq \beta' \leq \alpha' + 2\pi$ (Line 13). We then compute their penalties and examine whether they are better than the currently seen best refined query (Lines 14–16).

Algorithm 1 Answering Why-Not Questions: Special Case

INPUT: Original query $q = (loc, doc, \vec{w}, k_0, d_0 = [-\pi, \pi))$, Missing object m, Penalty option λ
OUTPUT: Best refined query $q' = (loc, doc, \vec{w}, k', d')$

1: compute $R(m, q, d_0)$ and record m's dominators in set S
2: $\theta_m \leftarrow$ **CalDirection**(q, m)
3: **for** each $o_i \in S$
4: $\quad\theta[i] \leftarrow$ **CalDirection**(q, o_i)
5: sort $\theta[i]$ in clockwise order w.r.t. θ_m
6: $d' \leftarrow d_0$, $k' \leftarrow R(m, q, d_0)$, $p_c \leftarrow \lambda$
7: **for** $r \leftarrow k_0$ **to** $R(m, q, d_0) - 1$ **do**
8: $\quad\Delta k \leftarrow r - k_0$
9: $\quad$**if** $\lambda \cdot \frac{\Delta k}{R(m,q,d_0) - k_0} \geq p_c$ **then**
10: $\quad\quad$**return** $q' \leftarrow (loc, doc, \vec{w}, k', d')$
11: $\quad$**for** $i \leftarrow 0$ **to** $r - 1$ **do**
12: $\quad\quad\alpha' \leftarrow \theta[i+1]$
13: $\quad\quad\beta' \leftarrow \alpha' + ((\theta[(R(m, q, d_0) - 1) - (r - i - 1)] - \theta[i + 1] + 2\pi)\%2\pi)$
14: $\quad\quad$compute the penalty p for the current candidate direction according to Eqn. (6)
15: $\quad\quad$**if** $p < p_c$ **then**
16: $\quad\quad\quad k' \leftarrow r$, $d' \leftarrow (\alpha', \beta')$, $p_c \leftarrow p$
17: **return** $q' \leftarrow (loc, doc, \vec{w}, k', d')$

Function CalDirection(q, o)
INPUT: Original query q, object o
OUTPUT: θ_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: $\quad$**if** $Y < 0$ **then return** $-\pi/2$
21: $\quad$**else return** $\pi/2$
22: $\theta_o \leftarrow \text{Arctan}(Y/X)$
23: **if** $X < 0$ and $Y \geq 0$ **then**
24: $\quad\theta_o \leftarrow \theta_o + \pi$
25: **if** $X < 0$ and $Y < 0$ **then**
26: $\quad\theta_o \leftarrow \theta_o - \pi$
27: **return** θ_o

Example 5. *Take Fig. 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 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 1 and compute their penalties to*

2. We use % to denote the modular operation throughout the paper.

check whether they are better than the known refined queries. For example, the candidate directions that give m a rank of 3 are: $(\theta[1], \theta[4])$, $(\theta[2], \theta[5])$ and $(\theta[3], \theta[6])$.

We next analyze the time complexity of the proposed algorithm.

Theorem 2. The time complexity of Algorithm 1 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 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.

5.1 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. (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 4, the bisector of the initial direction is fixed in the general case, *i.e.*, $\frac{\alpha_0+\beta_0}{2}$. According to Eqn. (4), the maximum possible modification of the direction is as follows:

$$\Delta d_{max} = \gamma \cdot \pi + (1-\gamma) \cdot Max\{\beta_0 - \alpha_0, 2\pi - (\beta_0 - \alpha_0)\} \quad (7)$$

This is obtained when both rotating the initial direction and changing the size of d_0 reach the maximum values, *i.e.*, π 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. 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.*, θ_m) to α_0 and β_0. We measure the "distance" as the length of the direction interval from β_0 to θ_m or from θ_m to α_0. In ②, θ_m is closer to β_0, since $(\theta_m - \beta_0 + 2\pi)\%2\pi$ is smaller than $(\alpha_0 - \theta_m + 2\pi)\%2\pi$. In ③, θ_m is closer to α_0.

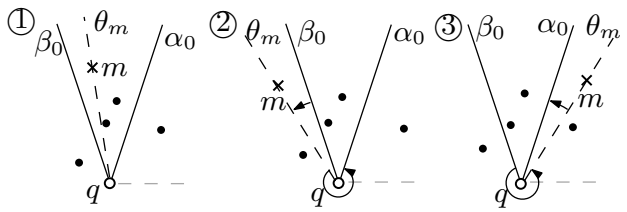


Fig. 4. Cases of the reason about m's missing

Next, we analyze the problem in these cases.

5.1.1 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 1, 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:

$$Penalty(q, q') = \lambda \cdot \frac{\Delta k}{R(m, q, d_0) - k_0} + (1-\lambda) \cdot \frac{\Delta d}{\gamma \cdot \pi + (1-\gamma) \cdot Max\{\beta_0 - \alpha_0, 2\pi - (\beta_0 - \alpha_0)\}} \quad (8)$$

Like the special case in Section 4, 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 3. Consider an initial query q with direction $d_0 = (\alpha_0, \beta_0)$ and a missing object m in d_0. Let θ_m be the angle of m *w.r.t.* q. A direction (α', β') 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 α' is a candidate start angle for the best refined query. Then, the possible refined end angle β' 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, $\Delta r_{[\alpha'', \beta']} - \Delta r_{[\alpha', \beta']} = |\frac{\alpha'' + \beta'}{2} - \frac{\alpha_0 + \beta_0}{2}| - |\frac{\alpha' + \beta'}{2} - \frac{\alpha_0 + \beta_0}{2}| = \frac{\alpha' - \alpha''}{2} < 0$. 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, $\Delta r_{[\alpha'', \beta'']} = |\frac{\theta_m - (\beta_0 - \alpha_0) + 2\beta_0 - \theta_m}{2} - \frac{\alpha_0 + \beta_0}{2}| = 0 \leq \Delta r_{[\alpha', \beta']}$. 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.

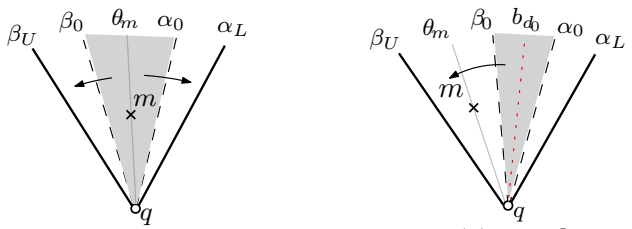
The proof of *(b)* is similar and hence omitted in the interest of space.

Theorem 3 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 **C**andidate **S**pace as $CS(d')$, *i.e.*, $CS(d') = [\theta_m - (\beta_0 - \alpha_0), \theta_m + (\beta_0 - \alpha_0)]$.

5.1.2 Case ②

Similar to Case ①, the search space for Case ② can be reduced by Theorem 4 to a smaller candidate space $CS(d') = [\alpha_0 - (\theta_m - \beta_0), \theta_m + (\beta_0 - \alpha_0)]$.

Theorem 4. Consider an initial query q with a direction $d_0 = (\alpha_0, \beta_0)$ and a missing object m outside d_0. Let θ_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 (α', β') can result in the best refined query if: *(i)* $\theta_m \in (\alpha', \beta')$; and *(ii)* $(\alpha', \beta') \subset [\alpha_0 - (\theta_m - \beta_0), \theta_m + (\beta_0 - \alpha_0)]$.



(a) Case ① (b) Case ②

Fig. 5. The candidate space of refined directions

Proof. We omit the proof since it is similar to that of Theorem 3.

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 4 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 Δk and avoid the denominator Δk_{max} being 0, the maximum modification on k is given as follows:

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

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

$$Penalty(q, q') = \lambda \cdot \frac{\Delta k}{max\{R(m, q, CS(d')) - k_0, 1\}} + (1 - \lambda) \cdot \frac{\Delta d}{\gamma \cdot \pi + (1 - \gamma) \cdot Max\{\beta_0 - \alpha_0, 2\pi - (\beta_0 - \alpha_0)\}} \quad (9)$$

5.1.3 Case ③

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

Example 6. *Fig. 5 illustrates the candidate space for refined directions, i.e., $CS(d') = [\alpha_L, \beta_U]$, in Case ① and ②. For Case ①, the lower bound α_L of the start angle is obtained by rotating the initial query direction clockwise until the end direction β_0 reaches θ_m, then the corresponding α_0 is α_L; the upper bound β_U of the end angle is obtained in a similar way by rotating d_0 counterclockwise until α_0 reaches θ_m. For Case ②, while β_U is also obtained similarly, α_L is actually the symmetrical angle of m's angle θ_m w.r.t. b_{d_0}, where b_{d_0} is the bisector of the initial query direction d_0, i.e., $\frac{\alpha_L + \theta_m}{2} = \frac{\alpha_0 + \beta_0}{2}$.*

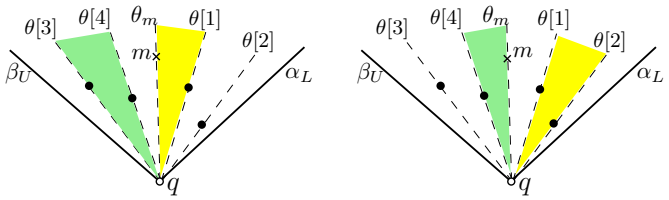
Algorithm 2 Answering Why-not Questions: General Case

INPUT: Original query $q = (loc, doc, \vec{w}, k_0, d_0)$,
Missing object m, Penalty Option λ, γ
OUTPUT: Best refined query $q' = (loc, doc, \vec{w}, k', d')$

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

5.2 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 2. First, we identify which case the problem falls into according to the relationship between θ_m and d_0 (Line 2). Then the candidate search space $CS(d')$, the maximum possible Δk, 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 16–35). Finally, we return the best one as the result (Line 37).



(a) Range 1 (b) Range 2

Fig. 6. Ranges of α and β that rank m at 2

Unlike the special case in Section 4, 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 α' and β'. See Fig 6 as an example. With either pair of $\alpha' \in (\theta[1], \theta_m], \beta' \in [\theta[4], \theta[3])$ 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 o_i in S as $\theta[i]$ and sort them clockwise *w.r.t* θ_m. Let t be the number of objects in S and in the direction $[\alpha_L, \theta_m]$. For a given k', the possible α' 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 θ_m. The corresponding β' range for an α' range can then be selected accordingly by ensuring that $k'-1$ dominators of m exist in $[\alpha', \beta']$. There are at most k' such α' and β' range pairs for a given k'. For example, in Fig. 6, 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 α' and β', with the objective function set as the corresponding penalty function (Line 31). Given an initial query q and a refined k', Δk is determined. Minimizing the penalty function is equivalent to minimizing Δd. Specifically, the linear programming problem of finding the optimal refined direction for a range pair of α' and β' that ranks the missing object m at a given rank k' can be modeled as follows:[3]

$$min \quad \Delta d = \gamma \cdot |\tfrac{\alpha'+\beta'}{2} - \tfrac{\alpha_0+\beta_0}{2}| + (1-\gamma) \cdot |(\beta' - \alpha') - (\beta_0 - \alpha_0)|$$

$$s.t. \begin{cases} \alpha' \in (\theta[i], \theta[i-1]] \\ \beta' \in [\theta[i-1] + (\theta[|S| - (k'-i) + 1] - \theta[i-1] + 2\pi)\%2\pi, \\ \quad \theta[i-1] + (\theta[|S| - (k'-i)] - \theta[i-1] + 2\pi)\%2\pi), \end{cases}$$

where $i \in \mathbb{Z}$ and $1 \leq i \leq min\{k', t+1\}$, and α_0 and β_0 are the start and end angles of the initial query's direction. Note that the β' range is computed in a way that ensures that it satisfies the constraint $\beta' \in [\alpha', \alpha' + 2\pi)$.

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

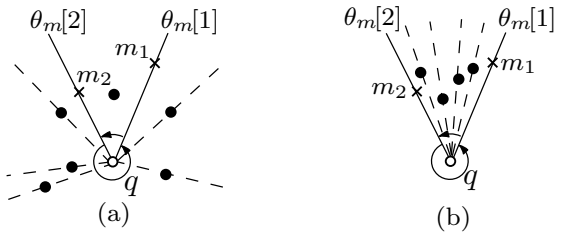


Fig. 7. An example of multiple missing objects

Example 7. *Fig. 6 shows an example, where two range pairs of α and β rank m at 2. The first pair is $\alpha' \in (\theta[1], \theta_m]$ and $\beta' \in [\theta[4], \theta[3])$ (Fig. 6(a)). To ensure that the refined β' is in $[\alpha', \alpha' + 2\pi)$, the range of β' is measured from θ_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 α' and β' in this range are then computed by solving the linear programming problem of minimizing Δd. It is computed similarly for the second range pair of α' and β' (Fig. 6(b)). By comparing these two results, we then find the best α' and β' that rank m at 2.*

We give the time complexity of Algorithm 2 in the next theorem.

Theorem 5. 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 2 has a time complexity of $O(DSKT(r) + r^2)$.

Proof. The proof is similar to that of Algorithm 1. The difference is that in Algorithm 2, we enumerate the possible k' from 1 to r, and for each k', there exist at most k' range pairs of refined α' and β'. 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 4 are also applicable to Algorithm 2.

6 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.*, $\forall m_i \in M$ ($\theta_{m_i} \in d'$). 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. 7(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. 7(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, *i.e.*, $[\theta_m[2], \theta_m[1]]$, is only 2, which incurs a smaller penalty for modifying k.

Theorem 6. 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 $O(SKT(r) + |M| \cdot r^2)$.

Optimizations: In addition to the optimizations that we proposed for a single missing object, we enumerate the sufficient directions in increasing order of the ranks of the worst ranked object among all missing objects in them, *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.

7 Empirical Study

This section evaluates the effectiveness and efficiency of the proposed algorithms.

7.1 Experimental Setup

7.1.1 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 [15] 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 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 CPU time.

7.1.2 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], [9], [11], [30]. 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 1 gives more details about the datasets.

TABLE 1
Dataset Information

Dataset	EURO	GN
Total # of objects	162,033	1,868,821
Total # of distinct words	35,315	222,407
Avg. # of words per object	18	4

7.1.3 Parameters

We evaluate the performance of our algorithms when varying different parameters. Table 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 γ in Eqn. (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.

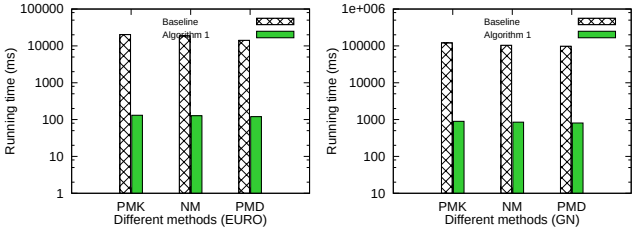
TABLE 2
Parameter Setting

Parameter	Setting
k_0	1, 3, **10**, 30, 100
# of keywords	2, **4**, 6, 8, 16
$\vec{w}$	$<0.1, 0.9>$,$<0.3, 0.7>$, $<$**0.5,0.5**$>$,$<0.7, 0.3>$,$<0.9, 0.1>$
$R(m, q, [-\pi, \pi))$	11, 31, **101**, 301, 1001
size of d_0	$\frac{\pi}{6}$, $\frac{\boldsymbol{\pi}}{\mathbf{3}}$, $\frac{\pi}{2}$, π, 2π
# of missing objects	**1**, 3, 10, 30

7.2 Experimental Results

7.2.1 Baseline vs. Algorithm 1

We first compare Algorithm 1 against the baseline method presented in Section 3.3. Note that the baseline method is workable for the special case only, since it is impossible to enumerate an infinite number of candidate directions in the general case. Thus, in this set of experiments, the initial query is a traditional spatial keyword top-10 query with no query direction. The candidate directions for the baseline algorithm are obtained according to Proposition 1. Fig. 8 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 1 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 1 is able to calculate the penalty of a candidate direction in constant time.



(a) Running Time (EURO) (b) Running Time (GN)

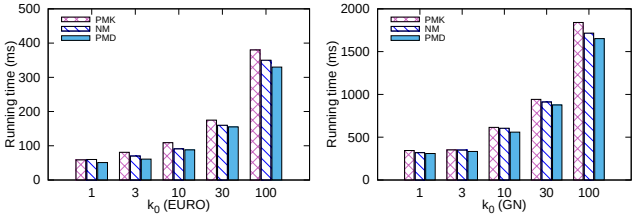
Fig. 8. Varying penalty options (Baseline vs. Algorithm 1)

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.

7.2.2 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 *w.r.t.* the initial query. Fig. 9 plots the results. Recall that the proposed algorithm consists of two phases, *i.e.*, *i)* computing the initial rank of the missing object and *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.

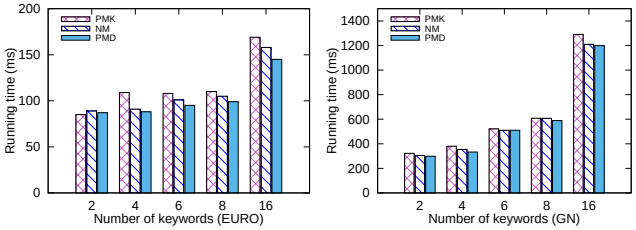


(a) Running Time (EURO) (b) Running Time (GN)

Fig. 9. Varying k_0

7.2.3 Varying the number of query keywords

We next evaluate the effect of different numbers of query keywords. Fig. 10 plots the results. Intuitively, the number of query keywords only affects the first phase of the algorithm, *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.



(a) Running Time (EURO) (b) Running Time (GN)

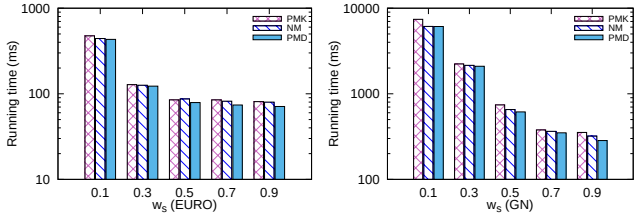
Fig. 10. Varying # query keywords

7.2.4 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 $\vec{w}$ also only affect the algorithm in the first phase. As we can see from Fig. 11, the query time decreases when w_s in $\vec{w}$ 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.

7.2.5 Varying the size of d_0

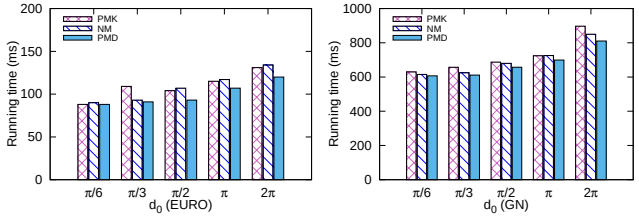
Next, we investigate the performance of the algorithm when varying the initial query direction d_0. In the experiments, queries with different initial direction sizes, from $\frac{\pi}{6}$ to 2π, are generated randomly. The average query time is shown in Fig. 12. A larger d_0 results in a larger candidate space $CS(d')$ for the refined query directions. As the processing of a direction-aware spatial keyword query consumes more time



(a) Running Time (EURO) (b) Running Time (GN)

Fig. 11. Varying w_s in $\vec{w}$

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 2π.



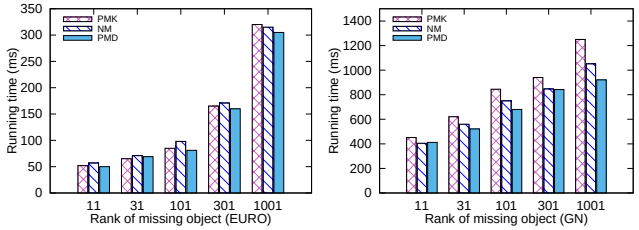
(a) Running Time (EURO) (b) Running Time (GN)

Fig. 12. Varying the size of d_0

7.2.6 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. 13 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.

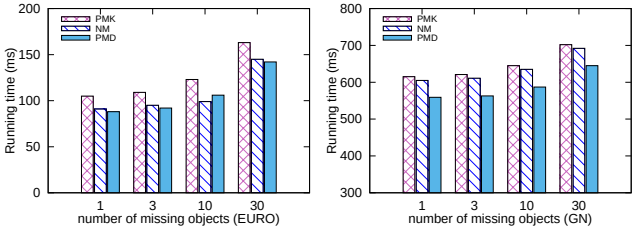


(a) Running Time (EURO) (b) Running Time (GN)

Fig. 13. Varying the initial rank of the missing object

7.2.7 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. 14 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.

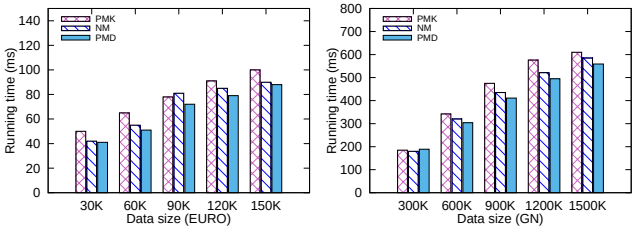


(a) Running Time (EURO) (b) Running Time (GN)

Fig. 14. Varying # missing objects

7.2.8 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. 15 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.



(a) Running Time (EURO) (b) Running Time (GN)

Fig. 15. Varying 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 λ only affects the second phase of the algorithms. Since we enumerate the possible refined k' settings in an increasing order, a larger λ can help stop the enumeration earlier. This explains the overall trend of the query time with different penalty option settings: *PMK*>*NM*>*PMD*.

8 Conclusions and Future Work

In this paper, we have studied the problem of answering why-not questions in the context of direction-aware spatial keyword top-k queries by refining users' query directions. We aim to minimally modify users' initial queries to revive their expected but missing objects. We have tackled the problem in two cases. Based on insightful problem analysis, we proved that the solution space is a finite set of candidates for the special case where no initial query direction is specified, and we provided a linear programming solution for the general case. We have also extended the proposed algorithms to support multiple missing objects. Extensive experiments with real datasets demonstrate that the proposed algorithms are scalable and are of superior performance compared to a baseline method under a wide range of system settings.

As for future work, we plan to investigate the refinement of the query location to make this line of work more complete. We also plan to study the relevant *why* questions to explain to users why some particular objects appear in a query result. Based on this, we would like 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.

Acknowledgments

This work is supported by HK-RGC Grants 12201615, 12244916, and 12200817. The work of Yafei Li is supported by NSFC Grant 61602420.

References

[1] S. S. Bhowmick, A. Sun, and B. Q. Truong, "Why Not, WINE?: Towards answering why-not questions in social image search," in *Proc. 21st ACM Int. Conf. on Multimedia*, pp. 917–926, 2013.

[2] P. Bouros, S. Ge, and N. Mamoulis, "Spatio-textual similarity joins," *Proc. VLDB Endowment*, vol. 6, no. 1, pp. 1–12, Nov. 2012.

[3] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial Keyword Querying," in *31st International Conference ER*, pp. 16–29, 2012.

[4] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *Proc. VLDB Endowment*, vol. 3, no. 1–2, pp. 373–384, Sep. 2010.

[5] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp. 373–384, 2011.

[6] A. Chapman and H. V. Jagadish, "Why not?," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp. 523–534, 2009.

[7] Lisi Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *Proc. VLDB Endowment*, vol. 6, no. 3, pp. 217–228, 2013.

[8] Lu Chen, Y. Gao, K. Wang, C. S. Jensen and G. Chen, "Answering why-not questions on metric probabilistic range queries," in *Proc. IEEE 32nd Int. Conf. Data Eng.*, pp. 767–778, 2016.

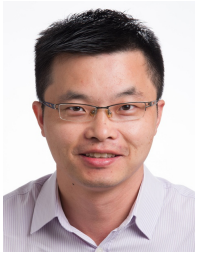
[9] Lei Chen, X. Lin, H. Hu, C. S. Jensen, and J. Xu, "Answering why-not questions on spatial keyword top-k queries," in *Proc. IEEE 31st Int. Conf. Data Eng.*, pp. 279–290, 2015.

[10] Lei Chen, Y. Li, J. Xu, and C. S. Jensen, "Direction-Aware Why-Not Spatial Keyword Top-k Queries," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, pp. 107–110, 2017 (short paper).

[11] Lei Chen, J. Xu, X. Lin, C. S. Jensen and H. Hu, "Answering why-not spatial keyword top-k queries via keyword adation," in *Proc. IEEE 32nd Int. Conf. Data Eng.*, pp. 697–708, 2016.
[12] Lei Chen, J. Xu, C. S. Jensen, and Y. Li, "YASK: A why-not question answering engine for spatial keyword query services," *Proc. VLDB Endowment*, vol. 9, no. 13, pp. 1501–1504, Sep. 2016.
[13] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient query processing in geographic web search engines," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp. 277–288, 2006.
[14] F. M. Choudhury, J. S. Culpepper, T. Sellis and X. Cao, "Maximizing bichromatic reverse spatial and textual k nearest neighbor queries," *Proc. VLDB Endowment*, vol. 9, no. 6, pp. 456–467, Jan. 2016.
[15] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 337–348, Aug. 2009.
[16] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. IEEE 24th Int. Conf. Data Eng.*, pp. 656–665, 2008.
[17] C. Manning, P. Raghavan, and H. Schutze, "Introduction to Information Retrieval," *Cambridge University Press*, 2008.
[18] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu, "SEAL: Spatio-textual similarity search," *Proc. VLDB Endowment*, vol. 5, no. 9, pp. 824–835, May 2012.
[19] Y. Gao, Q. Liu, G. Chen, B. Zheng, and L. Zhou, "Answering why-not questions on reverse top-k queries," *Proc. VLDB Endowment*, vol. 8, no. 7, pp. 738–749, Feb. 2015.
[20] X. Guo, Y. Ishikawa, Y. Xie, A. Wulamu, "Reverse direction-based surrounder queries for mobile recommendations," *World Wide Web*, vol. 20, no. 5, pp. 885–913, 2017.
[21] Z. He and E. Lo, "Answering why-not questions on top-k queries," in *Proc. IEEE 30th Int. Conf. Data Eng.*, pp. 750–761, 2012.
[22] M. Herschel and M. A. Hernández, "Explaining missing answers to SPJUA queries," *Proc. VLDB Endowment*, vol. 3, no. 1–2, pp. 185–196, Sep. 2010.
[23] J. Huang, T. Chen, A. Doan, and J. F. Naughton, "On the provenance of non-answers to queries over extracted data," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 736–747, Aug. 2008.
[24] M. S. Islam, R. Zhou, and C. Liu, "On answering why-not questions in reverse skyline queries," in *Proc. IEEE 29th Int. Conf. Data Eng.*, pp. 973–984, 2013.
[25] M.-J. Lee, D.-W. Choi, S. Kim, H.-M. Park, S. Choi, and C.-W. Chung, "The direction-constrained k nearest neighbor query - Dealing with spatio-directional objects," *GeoInformatica*, vol. 20, no. 3, pp. 471–502, 2016.
[26] G. Li, J. Feng, and J. Xu. "DESKS: Direction-aware spatial keyword search," in *Proc. IEEE 28th Int. Conf. Data Eng.*, pp. 474-485, 2012.
[27] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang, "IR-tree: an efficient index for geographic document search," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 585–599, Apr. 2011.
[28] F. Li, B. Yao, M. Tang, and M. Hadjieleftheriou, "Spatial approximate string search," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 6, pp. 1394–1409, Jun. 2013.
[29] X. Lin, J. Xu, and H. Hu, "Reverse keyword search for spatio-textual top-k queries in location-based services," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 11, pp. 3056–3069, Nov. 2015.
[30] J. Lu , Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp. 349–360, 2010.
[31] T. Lee, J. Park, S. Lee, S. Hwang, S. Elnikety, and Y. He, "Processing and optimizing main memory spatial-keyword queries," *Proc. VLDB Endowment*, vol. 9, no. 3, pp. 132–143, Nov. 2015.
[32] A. Motro, "Query generalization: A method for interpreting null answers," in *Proc. Int. Expert Database Workshop*, pp. 597–616, 1984.
[33] A. Motro, "SEAVE: A mechanism for verifying user presuppositions in query systems," *ACM Trans. Inf. Syst.*, vol. 4, no. 4, pp. 312–330, 1986.
[34] J. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *Proc.15th Int. Conf. Extending Database Technology*, pp. 168–179, 2012.
[35] J. Shi, D. Wu, and N. Mamoulis, "Top-k relevant semantic place retrieval on spatial RDF data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp. 1977–1990, 2016.
[36] Q. T. Tran and C. Chan, "How to ConQueR why-not questions," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pp. 15–26, 2010.
[37] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *The VLDB Journal*, vol. 21, no. 6, pp. 797–822, Dec. 2012.
[38] D. Wu, Y. Li, B. Choi, and J. Xu, "Social-aware top-k spatial keyword search," in *Proc. IEEE 15th Int. Conf. Mobile Data Management*, pp. 235–244,2014.
[39] X. Xie, X. Lin, J. Xu, and C. S. Jensen. "Reverse Keyword-based Location Search," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, pp. 375–386, 2017.
[40] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Proc. IEEE 25th Int. Conf. Data Eng.*, pp. 688–699, 2009.
[41] D. Zhang, B. C. Ooi, and A. K. H. Tung, "Locating mapped resources in web 2.0," in *Proc. IEEE 26th Int. Conf. Data Eng.*, pp. 521–532, 2010.

Lei Chen received his BEng degree from the College of Computer Science and Technology in South China University of Technology, Guangzhou, China, in 2012, and his PhD degree in Computer Science from Hong Kong Baptist University, Hong Kong, in 2016. He is currently a researcher at Huawei Noah's Ark Lab, Hong Kong. His research interests include spatial databases, query processing, and applied machine learning.

Yafei Li is an assistant professor in the School of Information Engineering, Zhengzhou University, Zhengzhou, China. He holds a visiting position in the database research group (http://www.comp.hkbu.edu.hk/∼db) at Hong Kong Baptist University. He received his PhD degree in Computer Science from Hong Kong Baptist University in 2015. His research interests include mobile and spatial data management, location-based services, and smart city computing.



Jianliang Xu is a professor in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China and his PhD degree in computer science from Hong Kong University of Science and Technology. He held visiting positions at Pennsylvania State University and Fudan University. His research interests include big data management, mobile computing, data security and privacy. He has published more than 150 technical papers in these areas. He has served as a program co-chair/vice chair for a number of major international conferences including IEEE ICDCS 2012, IEEE CPSNA 2015, and WAIM 2016. He is an Associate Editor of IEEE TKDE and PVLDB 2018.

Christian S. Jensen is an Obel professor of computer science at Aalborg University, Denmark. He was recently at Aarhus University for three years and at Google Inc. for one year. His research concerns data management and data-intensive systems, and its focus is on temporal and spatio-temporal data management. He has received several national and international awards for his research. He is an editor-in-chief of the ACM Transactions on Database Systems (TODS) and was an editor-in-chief of The VLDB Journal from 2008 to 2014. He is an ACM and an IEEE fellow, and he is a member of the Academia Europaea, the Royal Danish Academy of Sciences and Letters, the Danish Academy of Technical Sciences, and the EDBT Endowment, as well as a trustee emeritus of the VLDB Endowment.