# Answering Why-Not Questions on Spatial Keyword Top-$k$ Queries

Lei Chen[1] Xin Lin[2] Haibo Hu[1] Christian S. Jensen[3] Jianliang Xu[1]
[1]*Department of Computer Science, Hong Kong Baptist University, Hong Kong, China*
{lchen, haibo, xujl}@comp.hkbu.edu.hk
[2]*Shanghai Key Laboratory of Multidimensional Information Processing, East China Normal University, Shanghai, China*
xlin@cs.ecnu.edu.cn
[3]*Department of Computer Science, Aalborg University, Denmark*
csj@cs.aau.dk

***Abstract*—Large volumes of geo-tagged text objects are available on the web. 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. In this setting, users may wonder why some known object is unexpectedly missing from a result; and understanding why may aid users in retrieving better results. 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. We provide techniques that allow the revision of spatial keyword queries such that their results include one or more desired, but missing objects. In doing so, we adopt a query refinement approach to provide a basic algorithm that reduces the problem to a two-dimensional geometrical problem. To improve performance, we propose an index-based ranking estimation algorithm that prunes candidate results early. Extensive experimental results offer insight into design properties of the proposed techniques and suggest that they are efficient in terms of both running time and I/O cost.**

## I. Introduction

With the rapid development of mobile portable devices and location positioning technologies, the web is acquiring an increasingly prominent geographical flavor. Many web objects or points of interest, such as shops, hotels, and public services, are associated with both geographical tags and text descriptions. These attributes enable spatial keyword queries, such as finding nearby objects with the keyword "seafood." More specifically, a spatial keyword top-$k$ query takes a location and a set of keywords as arguments and retrieves $k$ objects that score the best according to a ranking function that takes into account these arguments [8]. However, users may not understand why desirable objects are unexpectedly missing from a result, and it may be difficult for users to ensure that the ranking function is configured so that the results include desired objects. It is therefore of high interest to be able to provide explanations of why desired objects are missing from a query result as well as suggestions for how to revise the query so that the desired objects are included in the result.

**[Example]** *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, 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?*

Several approaches may be taken to support such *why-not* questions, including *manipulation identification* [7], *database modification* [14], [15], and *query refinement* [13], [24]. Chapman and Jagadish [7] 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 [14], [15] study how to update a database so that SPJ queries and SPJUA (SPJ + Union + Aggregation) queries revive missing objects. Early work [24] using the query refinement approach studies how to revise an original query so that a missing object enters the result. He and Lo [13] 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, the existing solutions work for static datasets only, and they do not apply to spatial keyword queries where query locations are dynamic and precomputation based on spatial distance is infeasible.

In this paper, we adopt the query refinement approach and formally model the why-not spatial keyword problem. To efficiently answer a 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.

The paper's contributions are summarized as follows:

- We are the first to model the why-not spatial keyword query as a query refinement problem.
- We reduce the ranking update needed to include missing objects to a geometric problem, reduce the search space to a finite set of candidate vectors, and present an efficient query processing algorithm.

- We propose an index-based ranking estimation algorithm that prunes candidate weighting vectors and improves query processing performance.
- We extend the proposed algorithms to support multiple missing objects.
- We conduct extensive experiments to evaluate the efficiency of our proposed algorithms. The results show that our proposed solution 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.

The rest of this paper is organized as follows. Section II covers the background and related work. Section III describes some preliminaries and defines the why-not spatial keyword query. Section IV covers how to convert the ranking update to a geometric problem and proposes a basic query processing algorithm. Section V proposes an index-based optimization and Section VI extends the problem to handle multiple missing objects. The experimental results are presented in Section VII. Finally, we conclude in Section VIII.

## II. Related Work

To the best of our knowledge, there is no existing work on the processing of why-not spatial keyword queries. In the following, we cover studies on spatial keyword queries and why-not queries separately, and we show their differences in the context of why-not spatial keyword queries.

### A. 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, a number of efficient query processing techniques have been proposed. Early work proposed a hybrid index structure that integrates R*-trees and inverted files for the estimation of both spatial and textual similarities [29]. Martins *et al*. [21] suggest computing text relevancy and location proximity independently and then to combine the two. More recently, the IR-tree, a hybrid index that estimates the bounds of spatial distance and textual similarity at the same time, was proposed [8]. Another hybrid index, the IR$^2$-tree, has also been proposed in [10]. This index combines an R-tree with superimposed text signatures. However, it is applicable only when the keywords serve as a Boolean filter. Rocha-Junior and Nørvåg [22] investigate the spatial keyword query in road networks. A comprehensive experimental evaluation of different spatial keyword query indexing and query processing techniques has also appeared [5].

Different variants of the spatial keyword query have been considered. Chen *et al*. [6] study a query that retrieves web pages containing query keywords and whose page footprints intersect a query footprint. A recently proposed $m$CK query retrieves $m$ objects within a minimum diameter that match the given keywords. The bR*-tree and the virtual bR*-tree [27], [28], which augment each node with a bitmap and MBRs for keywords, have been proposed for this query. Cao *et al*. [3] introduce a query that retrieves the top-$k$ spatial web objects ranked according to both prestige-based relevance and location proximity. Another study [4] proposes a variant that retrieves a group of spatial web objects whose keywords cover the query's keywords and that are the nearest to the query location and have the lowest inter-object distances. More recently, spatial keyword similarity search in regions of interest was studied [11]. Other authors consider a direction-aware spatial keyword query that finds $k$ nearest neighbors in the search direction that cover all query keywords [17]. Another study [18] considers a spatial approximate string query that is a range query augmented with a string similarity predicate. Next, a study [2] aims to identify pairs of objects from a spatio-textual database that are both spatially close and textually similar and a study [25] integrates social influence into traditional spatial keyword search to improve answer quality. However, none of the works covered here address the why-not spatial keyword query problem.

### B. Why-Not Query Processing

To improve the usability of database systems, the why-not problem was first introduced by Chapman and Jagadish [7]. Existing approaches can be classified into three categories. Chapman and Jagadish used *manipulation identification* to identify operations that filter out a missing object. Other studies [14], [15] adopt a *database modification* approach that updates an original database so that missing objects become part of SPJ and SPJUA query results. Tran and Chan [24] propose to retrieve missing objects through *query refinement*. He and Lo [13] employ query refinement to answer why-not questions on top-$k$ preference queries. They aim to modify the original query with the minimum penalty. More recently, two studies [1], [16] consider how to answer why-not questions in the contexts of social image search and reverse skyline queries. However, the why-not problem has not been studied for spatial keyword queries.

## III. 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.

### A. 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 [8] as follows:

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

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 $\vec{w}$=$\langle ws, wt\rangle$, 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 [20], such as the language model, cosine similarity, or BM25, and is also normalized. We adopt the language model in this paper, so the range of $TSim(o,q)$ is (0,1). In the ranking function, the higher the score computed by Eqn. 1, the higher the rank of the corresponding object. Objects that rank higher than $o$ are called $o$'s *dominator*s under the given weighting vector $\vec{w}$, and we define the rank of an object as follows:

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

With the definition of ranking, 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 $k$ objects in $\mathcal{D}$ that maximize the scoring function in Eqn. 1, or in terms of ranking, it returns the object set $\{o \mid R(o,q,\vec{w}) \leq k\}$.

### B. Why-Not Spatial Keyword Query

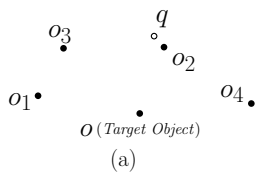
When a user issues a spatial keyword top-$k$ query $q = (loc, doc, k_0, \vec{w_0})$, the user may observe that one or more objects that were expected to be in the result are missing. The user may then pose a *why-not* query with a set of missing objects $M = \{o_1, o_2, ..., o_j\}$, asking the system to identify and process a refined spatial keyword query $q' = (loc, doc, k', \vec{w'})$ that has a result that contains the missing objects. In devising the refined query, we consider the modification of the parameters $k$ and $\vec{w}$ 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 [13], 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 $\Delta w = \|\vec{w'} - \vec{w_0}\|_2$. Based on this, the penalty (*i.e.*, cost) of a refined query from $q$ is defined as follows:

$$Penalty(k', \vec{w'}) = \lambda \cdot \frac{\Delta k}{R(o,q,\vec{w_0}) - k_0} + (1-\lambda) \cdot \frac{\Delta w}{\sqrt{1 + w s_0^2 + w t_0^2}}, \quad (3)$$

where $\lambda \in (0,1)$ is a user preference of the modification on $k$ and $\vec{w}$ from the initial query. In the best refined query with weighting vector $\vec{w'}$, if $R(o,q,\vec{w'}) > k_0$, $k'$ should be equal to $R(o,q,\vec{w'})$ to achieve the lowest penalty; otherwise, $k'$ does not need to be modified. So $\Delta k = max(0, R(o,q,\vec{w'}) - k_0)$. We normalize $\Delta k$ and $\Delta w$ by dividing them by $R(o,q,\vec{w_0}) - k_0$ and $\sqrt{1 + w s_0^2 + w t_0^2}$, respectively. As proved in [13], $\Delta k$ in the best refined query is no larger than $R(o,q,\vec{w_0}) - k_0$, and $\Delta w$ is no larger than $\sqrt{1 + w s_0^2 + w t_0^2}$.

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

*Definition 2:* **Why-Not Spatial Keyword Query**. Given an object set $\mathcal{D}$, a missing object set $M \subset \mathcal{D}$, an original spatial keyword query $q$: $(loc, doc, k_0, \vec{w_0})$, the *why-not spatial keyword query* returns the refined query with the lowest penalty according to Eqn. 3, which includes all objects in $M$ in the query result.




| | $1 - SDist(o,q)$ | $TSim(o,q)$ |
|---|---|---|
| $o$ | 0.5 | 0.4 |
| $o_1$ | 0.25 | 0.5 |
| $o_2$ | 0.9 | 0.33 |
| $o_3$ | 0.55 | 0.25 |
| $o_4$ | 0.4 | 0.9 |

(b)

Fig. 1. An Example of Why-not Spatial Keyword Query

| Refined Query | $\Delta k$ | $\Delta w$ | Penalty |
|---|---|---|---|
| $q'_1(3, \langle 0.5, 0.5 \rangle)$ | 2 | 0.00 | 0.50 |
| $q'_2(3, \langle 0.4, 0.6 \rangle)$ | 2 | 0.14 | 0.56 |
| $q'_3(4, \langle 0.1, 0.9 \rangle)$ | 3 | 0.57 | 1.21 |

TABLE I. AN EXAMPLE OF REFINED QUERIES ($\lambda = 0.5$)

Fig. 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 $\mathcal{D}$. In the original query, $k_0 = 1$, $w_0 = \langle 0.5, 0.5 \rangle$. Object $o$ has $R(o,q,\vec{w_0}) = 3$, so it is missing from the top-1 result. Table I 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 $\sqrt{1 + w s_0^2 + w t_0^2} = 1.22$. 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.

### C. 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 $\mathcal{D}$ become two constant values. That is, each object can be represented by a vector $\vec{v} = \langle 1 - SDist(o,q), TSim(o,q) \rangle$, and the scoring function becomes $ST(o,q,\vec{w}) = \vec{v} \cdot \vec{w}$. 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 [13] as follows (for now we only consider a single missing object). We first calculate the vector $\vec{v}$ for each object in $\mathcal{D}$ and find the set $\mathcal{I}$ of objects in $\mathcal{D}$ that are *incomparable* with the missing object $o$ using an algorithm introduced in [23], 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 $\mathcal{S}$ containing the initial weighting vector $\vec{w_0}$ and the weighting vector $\vec{w_i}$ for each incomparable object $i$ that satisfies $\vec{v_i} \cdot \vec{w_i} = \vec{v_o} \cdot \vec{w_i}$. Finally, for each weighting vector $\vec{w_i}$ in $\mathcal{S}$, we issue a top-$k$ query to obtain the ranking of the missing object under $\vec{w_i}$. 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.

## IV. PROPERTIES AND BASIC ALGORITHM

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

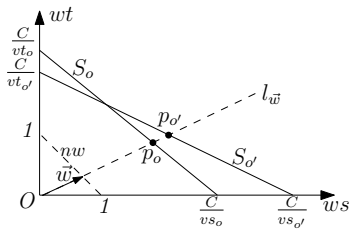



Fig. 2. An Example of Fixed-Score Segment

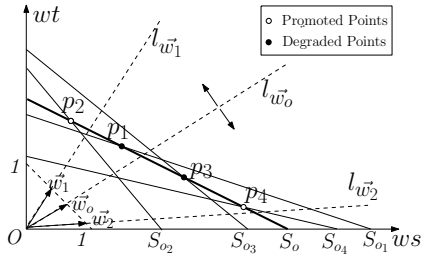



Fig. 3. An Example of Weighting Vector Rotation and Ranking Updates

TABLE II. SUMMARY OF NOTATIONS

| Notation | Meaning |
|---|---|
| $R(o, q, \vec{w})$ | The ranking of object $o$ under query $q$ and weighting vector $\vec{w}$. |
| $[\vec{w_1}, \vec{w_2}]$ | The angle interval from $\vec{w_1}$ to $\vec{w_2}$. |
| $l_{\vec{w}}$ | The line of weighting vector $\vec{w}$. |
| $p(l_1, l_2)$ | The intersection point of lines (or segments) $l_1$ and $l_2$. |
| $S_o$ | The fixed-score segment of object $o$. |
| $S_o^s$ or $S_o^t$ | The intersection point between segment $S$ and the $ws$-axis or $wt$-axis. |

We first use the transformation introduced by Chester *et al*. [9] to project the objects in $\mathcal{D}$ 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 II summarizes notations used in the following.

### A. 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 $q$ are constant. As such, we define a *fixed-score segment* of $o$ as follows.

*Definition 3:* **Fixed-score Segment**. For an object $o$, with $vs_o = 1 - SDist(o, q)$, $vt_o = TSim(o, q)$, and $C$ being a randomly selected positive real number, the fixed-score segment of $o$ in the 2D plane "$ws - wt$" is defined as the segment with end points $(0, \frac{C}{vt_o})$ and $(\frac{C}{vs_o}, 0)$.

Fig. 2 shows an example of an object $o$'s fixed-score segment (labeled by $S_o$). The meaning of $S_o$ is that for any point $p_o = (ws_o, wt_o)$ on $S_o$, it holds that $ws_o \cdot vs_o + wt_o \cdot vt_o = C$. Since $vs$ and $vt$ of all objects in $\mathcal{D}$ are constant after the user specifies the initial query, all objects can be represented by fixed-score segments. Further, the weighting vector $\vec{w}$ in the query can also be projected into the "$ws-wt$" plane. According to the definition of $\vec{w}$, 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 $S_o$ and $S_{o'}$, the comparison relationship between the similarity scores of the two corresponding objects under weighting vector $\vec{w}$, *i.e.*, $ST(o, q, \vec{w})$ and $ST(o', q, \vec{w})$, can be summarized in Theorem 1.

*Theorem 1:* Consider a weighting vector $\vec{w}$ in a query and two objects' fixed-score segments $S_o$ and $S_{o'}$ in the "$ws-wt$" plane (see Fig. 2). Let $l_{\vec{w}}$ denote the line of $\vec{w}$ and let $p_o$ ($p_{o'}$) denote the intersections of $l_{\vec{w}}$ and $S_o$ ($S_{o'}$). It holds that $ST(o, q, \vec{w}) > ST(o', q, \vec{w})$ 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 $\vec{w} = \langle w_s, w_t \rangle$. According to the property of fixed-score segments, $ws_o \cdot vs_o + wt_o \cdot vt_o = ws_{o'} \cdot vs_{o'} + wt_{o'} \cdot vt_{o'} = C$. Since $p_o$ and $p_{o'}$ are located on the line of $\vec{w}$, both $\frac{w_s}{ws_o} = \frac{w_t}{wt_o} = \frac{|\vec{w}|}{|Op_o|}$ and $\frac{w_s}{ws_{o'}} = \frac{w_t}{wt_{o'}} = \frac{|\vec{w}|}{|Op_{o'}|}$ hold. Hence, $ST(o, q, \vec{w}) = w_s \cdot vs_o + w_t \cdot vt_o = \frac{|\vec{w}|}{|Op_o|} \cdot (ws_o \cdot vs_o + wt_o \cdot vt_o) = \frac{|\vec{w}|}{|Op_o|} \cdot C$. Similarly, $ST(o', q, \vec{w}) = \frac{|\vec{w}|}{|Op_{o'}|} \cdot C$. As we assume $|Op_o| < |Op_{o'}|$, it holds that $ST(o, q, \vec{w}) > ST(o', q, \vec{w})$. □

If the condition $|Op_o| < |Op_{o'}|$ in Theorem 1 holds, we also say $S_o$ is closer to the origin than $S_{o'}$ in the $\vec{w}$ direction. By Theorem 1, we obtain the following proposition.

*Proposition 1:* 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 1 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 $\mathcal{D}$. 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.

### B. Query Modification and Ranking Updates

Proposition 1 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 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 $l_{\vec{w}}$ changes its intersection points with the fixed-score segments, which means that the rankings of objects also change. In Fig. 3, after $l_{\vec{w}}$ 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 $l_{\vec{w}}$ 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 $l_{\vec{w}}$ encounters $p_3$, the intersection point of $S_o$ and $S_{o_3}$, the ranking of $o$ should be decremented by 1, while after $l_{\vec{w}}$ 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 $l_{\vec{w}}$ 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 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, $l_{\vec{w}}$ 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, $p_1$ 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 $[l_{\vec{w_0}}, l_{\vec{w_1}}]$. Let the numbers of promoted and degraded points be $n_p$ and $n_d$, respectively. Thus, the ranking of $o$ under weighting vector $l_{\vec{w_1}}$ is $n - n_p + n_d$. For example, in Fig. 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 2:* 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. □

### C. Basic Algorithm for Why-Not Spatial Keyword Query

Theorem 2 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.*, [8]), 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 $\mathcal{D}$ 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. 1) We compute $R(o, q, \vec{w_0})$ (=3) and find all dominators of $o$ under $\vec{w_0}$ ($\{o_2, o_4\}$). 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.

## V. 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 V-A introduces the BIR-tree, and Section V-B 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 V-C describes how to compute the upper and lower bounds using a BIR-tree.

### A. 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 [8] and Fig. 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]

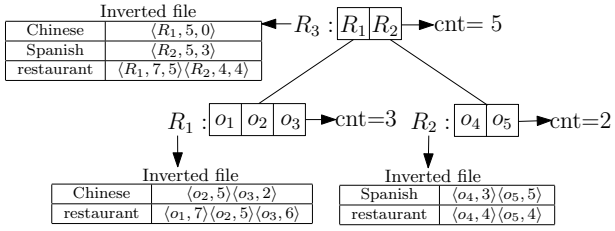



Fig. 4. Structure of a BIR-Tree

### B. 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 2, 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 $o$ 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 $p(l_{\vec{w_0}}, S_o)$. For points located exactly in the same place, the number is counted multiple times. As discussed in Section IV-B, $R(o, q, \vec{w_e})$=$R(o, q, \vec{w_0}) - PP(e) + DP(e)$. 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 $\check{DP}(N, e)$ (Section V-C provides the details). By these bounds, we get the upper bound $\hat{pn}(e)$ and lower bound $\check{pn}(e)$ on the penalty value (Lines 10–12). If the lower bound $\check{pn}(e)$ 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 $\check{pn}(e)$ 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

[1]The literature (*e.g.*, [20]) explains how to compute the weight of a keyword from keyword frequencies.

**Algorithm 1** Optimized Why-Not Spatial Keyword Query Processing Algorithm by Estimation

INPUT: BIR-tree $\mathcal{T}$, original query $q = (loc, doc, k, \vec{w_0})$, missing object $o$
OUTPUT: Best refined query $q_b = (loc, doc, k_b, \vec{w_b})$

1: determine $R(o, q, \vec{w_0})$ 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 $\vec{w_0}$ // the threshold recording the minimum penalty
8: **for** each element $e$ in $Pro$
9: $PP(e) \leftarrow$ # promoted points between $e$ and $p(l_{\vec{w_0}}, S_o)$
10: $\hat{R}(e) \leftarrow R(o, q, \vec{w_0})$ - $PP(e)$ + $\hat{DP}(\mathcal{T}.root, e)$ // Section V-C
11: $\check{R}(e) \leftarrow R(o, q, \vec{w_0})$- $PP(e)$+$\check{DP}(\mathcal{T}.root, e)$ // Section V-C
12: use $\hat{R}(e)$ and $\check{R}(e)$ to compute the lower bounds and upper bounds of penalty of $e$, *i.e.*, $\check{pn}(e)$ and $\hat{pn}(e)$
13: **if** $\check{pn}(e) > TH$ **then** prune $e$ from $Pro$
14: $\mathcal{Q} \leftarrow$ an empty queue
15: insert $\mathcal{T}.root$ into $\mathcal{Q}$
16: **while** $\mathcal{Q}$ is not empty **do**
17: $N \leftarrow$ **Dequeue**($\mathcal{Q}$)
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: $\hat{DP}' \leftarrow 0$
22: $\check{DP}' \leftarrow 0$
23: **for** each child $c$ of $N$ **do**
24: $\hat{DP}' \leftarrow \hat{DP}' + \hat{DP}(c, e)$
25: $\check{DP}' \leftarrow \check{DP}' + \check{DP}(c, e)$
26: $\hat{R}(e) \leftarrow \hat{R}(e) - (\hat{DP}(N, e) - \hat{DP}')$
27: $\check{R}(e) \leftarrow \check{R}(e) + (\check{DP}' - \check{DP}(N, e))$
28: use $\hat{R}(e)$ and $\check{R}(e)$ to compute $\check{pn}(e)$ and $\hat{pn}(e)$
29: **if** $\hat{pn}(e) < TH$ **then** $TH \leftarrow \hat{pn}(e)$
30: **for** each existing promoted point $e$ in $N$'s degraded part
31: **if** $\check{pn}(e) > TH$ **then** prune $e$ from $Pro$
32: insert each child $c$ of $N$ into $Q$
33: **for** each remaining promoted point $e$
34: $R(o, q, \vec{e}) \leftarrow \check{R}(e)$
35: compute $pn(e)$ by $R(o, q, \vec{e})$
36: $b \leftarrow$ the remaining promoted point with the minimum penalty
37: **if** $pn(b) <$ penalty of $\vec{w_0}$ **then** return $(loc, doc, R(o, q, \vec{w_b}), \vec{w_b})$
38: **else return** $(loc, doc, R(o, q, \vec{w_0}), \vec{w_0})$

(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 (Line 38).

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 $p(l_{\vec{w_0}}, S_o)$. 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 $\mathcal{D}$. The upper and lower bounds of $DP(e)$ are initialized to $\hat{DP}(\mathcal{T}.root, e)$ and $\check{DP}(\mathcal{T}.root, e)$, 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 $\sum \check{DP}(c, e) - \check{DP}(N, e)$, where $c$ denotes a child of $N$ (Lines 21–27). In Section V-C3, we prove that a promoted point can tighten the bounds only if it is located in the *degraded part* of the node.

### C. Derivation of $\hat{DP}(N, e)$ and $\check{DP}(N, e)$

As mentioned, the estimations of $\hat{DP}(N, e)$ and $\check{DP}(N, e)$ 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 $\check{DP}(N, e)$.

*1) 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 5:* **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 ($\frac{C}{vs_o}$ *vs.* $\frac{C}{vs_{o'}}$ and $\frac{C}{vt_o}$ *vs.* $\frac{C}{vt_{o'}}$):

1) If $\frac{C}{vt_{o'}} < \frac{C}{vt_o}$ and $\frac{C}{vs_{o'}} < \frac{C}{vs_o}$, $o'$ always ranks higher than $o$ and $o'$ is a *dominator* of $o$ under all candidate weighting vectors (denoted by $Dom(o)$);
2) If $\frac{C}{vt_{o'}} \geq \frac{C}{vt_o}$ and $\frac{C}{vs_{o'}} \geq \frac{C}{vs_o}$, $o'$ never ranks higher than $o$ and $o'$ is a *dominatee* under all candidate weighting vectors (denoted by $DomBy(o)$) ;
3) If $\frac{C}{vt_{o'}} \geq \frac{C}{vt_o}$ and $\frac{C}{vs_{o'}} < \frac{C}{vs_o}$, $S_o$ and $S_{o'}$ have an intersection point, and $o'$ is called a *steeper object* of $o$ (denoted by $Stp(o)$);
4) If $\frac{C}{vt_{o'}} < \frac{C}{vt_o}$ and $\frac{C}{vs_{o'}} \geq \frac{C}{vs_o}$, $S_o$ and $S_{o'}$ also have an intersection point, and $o'$ is called a *gentler object* of $o$ (denoted by $Gent(o)$).

Take Fig. 3 as an example. Here, $o_3$ is in $DomBy(o_2)$, while $o_2$ is in $Dom(o_3)$. Both $o_3$ and $o_2$ are in $Stp(o)$, while both $o_4$ and $o_1$ are in $Gent(o)$. As mentioned above, only the fixed-score segments of $Stp(o)$ and $Gent(o)$ 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 1:* 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 $p(l_{\vec{w}_0}, S_o)$; or 2) $o'$ is a gentler object of $o$ and $p(S_{o'}, S_o)$ is to the right of the intersection $p(l_{\vec{w}_0}, S_o)$. 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 $p(l_{\vec{w}_0}, S_o)$; or 2) $o'$ is a gentler object of $o$ and $p(S_{o'}, S_o)$ is to the left of the intersection $p(l_{\vec{w}_0}, S_o)$.

The proof is straightforward and thus omitted. In the example of Fig. 3, if the initial weighting vector is $\vec{w}_0$, $S_{o_1}$ and $S_{o_3}$ produce promoted points on $S_o$, while $S_{o_2}$ and $S_{o_4}$ produce degraded points.

*2) 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 $\hat{S}Dist(N,q)$) and minimum distance (denoted by $\check{S}Dist(N,q)$) 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 ($\hat{T}Sim(N,q)$) and lower bound ($\check{T}Sim(N,q)$) of textual similarities between the objects in $N$ and a query $q$. Let $\hat{vs}_N = 1 - \check{S}Dist(N,q)$, $\check{vs}_N = 1 - \hat{S}Dist(N,q)$, $\hat{vt}_N = \hat{T}Sim(N,q)$, and $\check{vt}_N = \check{T}Sim(N,q)$. We can draw two boundary segments $\hat{S}_N$ and $\check{S}_N$ for each node $N$ and $q$, where $\hat{S}_N$ is the segment with ends $(\frac{C}{\hat{vs}_N}, 0)$ and $(0, \frac{C}{vt_N})$, and $\check{S}_N$ is the segment with ends $(\frac{C}{\check{vs}_N}, 0)$ and $(0, \frac{C}{\hat{vt}_N})$. The region enclosed by the $vt$-axis, $\hat{S}_N$, the $vs$-axis, and $\check{S}_N$ is called the *boundary region* of $N$ and

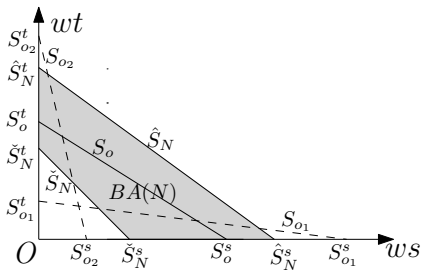



Fig. 5. An Example of the Boundary Region of a Node $N$

TABLE III. RELATIONS BETWEEN NODE $N$ AND OBJECT $o$

| $S^t_o$ \ $S^s_o$ | $(\hat{S}^s_N, \infty)$ | $[\check{S}^s_N, \hat{S}^s_N]$ | $(0, \check{S}^s_N)$ |
|---|---|---|---|
| $(\hat{S}^t_N, \infty)$ | ♣[1] | ♣♢[2] | ♢[3] |
| $[\check{S}^t_N, \hat{S}^t_N]$ | ♡♣[4] | ♡♢♣♠[5] | ♢♠[6] |
| $(0, \check{S}^t_N)$ | ♡[7] | ♡♠[8] | ♠[9] |
| ♡ = $Stp(o)$, ♢ = $Gent(o)$, ♣ = $Dom(o)$, ♠ = $DomBy(o)$ | | | |

is denoted by $BA(N)$ (see the shaded region in Fig. 5). The following theorem establishes a desirable property of $BA(N)$.

*Theorem 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. 5 illustrates the theorem. Since $\check{S}Dist(N,q)$ is the lower bound of the distance from the objects in $N$ to $q$, for any object $o$ in $N$, $SDist(o,q) \geq \check{S}Dist(N,q)$. So, $vs_o \leq \hat{vs}_N$, from which $\frac{C}{vs_o} \geq \frac{C}{\hat{vs}_N}$ follows. Similarly, $\frac{C}{vs_o} \leq \frac{C}{\check{vs}_N}$. This means that one end of $S_o$ is on the segment from $(0, \frac{C}{\hat{vs}_N})$ to $(0, \frac{C}{\check{vs}_N})$, 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 $BA(N)$. □

Next, the relation between a node $N$ and an object not in $N$ can be divide into nine cases, according to the relation between the intercepts of $S_o$ and the edges of $BA(N)$ on the $ws$-axis and the $wt$-axis. For ease of presentation, let $S^s_o$ ($S^t_o$) 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 $S^s_o \in (\hat{S}^s_N, \infty)$ and $S^t_o \in (0, \check{S}^t_N)$ hold, all objects in $N$ are steeper objects of $o$ (like $N$ and $o_1$ in Fig. 5); likewise, if both $S^t_o \in (\hat{S}^t_N, \infty)$ and $S^s_o \in (0, \check{S}^s_N)$ hold, all objects in $N$ are gentler objects of $o$ (like $N$ and $o_2$ in Fig. 5). Other relations between $N$ and $o$ are summarized in Table III, and we number the cases through 1 to 9 for latter reference. The column and row headers indicate the intervals that $S^s_o$ and $S^t_o$ belong to. The symbols used are explained in the last row. For example, ♡ means that the objects in $N$ belong $Stp(o)$. If $S^s_o \in [\check{S}^s_N, \hat{S}^s_N]$ and $S^t_o \in (\hat{S}^t_N, \infty)$ hold, symbols ♣♢ apply meaning that the objects in $N$ belong to $Dom(o)$ **or** to $Gent(o)$.

*3) Estimation of $\hat{DP}(N,e)$ and $\check{DP}(N,e)$:* 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 $p(l_{\vec{w}_0}, S_o)$, *i.e.*, $DP(N,e)$. We proceed

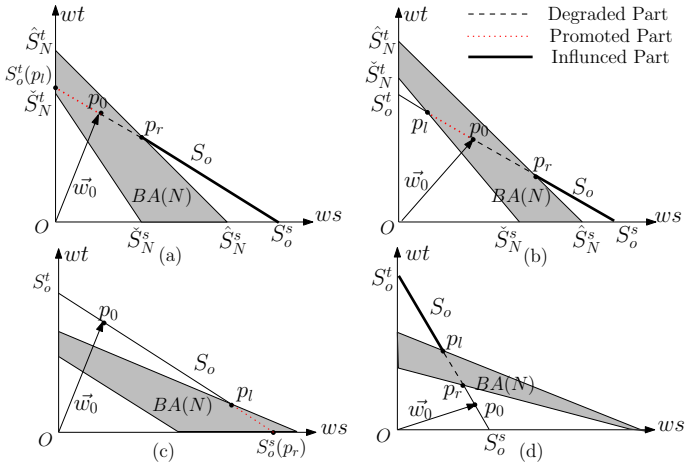



Fig. 6. An Example of # Degraded Point Estimation

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,

$$N.cnt = |N.Dom(o)|+|N.DomBy(o)|+|N.Gent(o)|+|N.Stp(o)|. \quad (4)$$

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

$$|N.Gent(o)| + |N.Stp(o)| = PP(N) + DP(N), \quad (5)$$

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. 5 into Eqn. 4, the number of degraded points on $S_o$ produced by the objects in $N$'s subtree can be expressed as:

$$DP(N) = N.cnt - |N.Dom(o)| - |N.DomBy(o)| - PP(N). \quad (6)$$

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

For relation 1 in Table III, all objects are in $Dom(o)$, *i.e.*, $N.Gent(o) = \emptyset$ and $N.Stp(o) = \emptyset$. According to Eqn. 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. 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. 6, the number of degraded points produced by the objects in such a node $N$ is:

$$DP(N) = N.cnt - |N.Dom(o)| - PP(N). \quad (7)$$

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 6) as the upper bound of $PP(N)$. According to Eqn. 7, the lower bound of $DP(N)$ can be estimated as:

$$\check{DP}(N) = \max\{0, N.cnt - PP(S^+_{o,N}) - |Dom(o)|\}. \quad (8)$$

In the other extreme, we assign 0 as the lower bound of $PP(N)$ and $|N.Dom(o)|$. By Eqn. 7, we can derive the upper bound of $DP(N)$ by the following equation:

$$\hat{DP}(N) = N.cnt. \quad (9)$$

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. 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. 6,

$$DP(N) = N.cnt - |N.DomBy(o)| - PP(N), \quad (10)$$

where we only know $PP(N) \in [0, PP(S^+_{o,N})]$, but not $|D.DomBy(o)|$. We therefore set $\check{DP}(N)$ 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. 6. For example, Fig. 6(a) corresponds to relation 4, *i.e.*, $S^s_o \in (\hat{S}^s_N, \infty)$ and $S^t_o \in [\check{S}^t_N, \hat{S}^t_N]$. According to Table III, 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 $\overline{p_l p_r}$, where $\overline{p_l p_r}$ 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 $p(l_{\vec{w}_0}, S_o)$ as $p_0$. According to Lemma 1, if $p(S_o, S_{o'})$ lies to the left of $p_0$ (*i.e.*, on the subsegment $\overline{p_l p_0}$), it must be a promoted point; otherwise, if $p(S_o, S_{o'})$ lies on the subsegment $\overline{p_0 p_r}$, it must be a degraded point. We call $\overline{p_l p_0}$ the *promoted part* of $S_o$ and $\overline{p_0 p_r}$ the *degraded part* of $S_o$.

*Definition 6:* **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. 6 illustrates the promoted and degraded parts, which are denoted by dotted and dashed segments, respectively. If $S^s_o \in (\hat{S}^s_N, \infty)$, either $S^t_o \in [\check{S}^t_N, \hat{S}^t_N]$ (see Fig. 6(a)) or $S^t_o \in (\hat{S}^t_N, \infty)$ (see Fig. 6(b)), the promoted part is to the left of $p_0$, and the degraded part is to the right. Conversely, if $S^t_o \in (\hat{S}^t_N, \infty)$, either $S^s_o \in [\check{S}^s_N, \hat{S}^s_N]$ (see Fig. 6(c)) or $S^s_o \in (\hat{S}^s_N, \infty)$ (see Fig. 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. 6(d)) or degraded part (see Fig. 6(c))

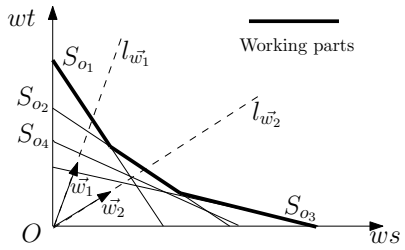



Fig. 7. An Example of Multiple Missing Objects

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. 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. 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:

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

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

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 $[0, N.cnt]$, 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).

## VI. 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. 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 $\vec{w}_1$, $o_1$ is the lowest-ranked missing object because the intersection of $S_{o_1}$ and the weighting vector line $l_{\vec{w}_1}$ is the farthest from the origin. Similarly, if $\vec{w}_2$ is the weighing vector, $o_2$ is the lowest-ranked missing object. Given an arbitrary weighting vector $\vec{w}$, we define the intersection of the line $l_{\vec{w}}$ and the farthest fixed-score segment as a *working point*. The working points 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. 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.

**Algorithm 2** Identification of Working Parts

```
INPUT: Missing object set M
OUTPUT: Working parts of missing objects WP
1:  WP ← ∅
2:  intersect the fixed-score segments of the objects in M with the wt and ws axises
3:  IS ← the ws-intercept set (in ascending order)
4:  denote the ith element in IS by is_i
5:  denote the fixed-score segment producing the intercept is_i by S(is_i)
6:  n ← |M|
7:  sp ← is_n
8:  while true do
9:    for i =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)^t ← the intersection of S(is_n) and the wt-axis
13:     insert sub-segment (sp, S(is_n)^t) into WP
14:     break
15:   else
16:     p_c ← the intersection nearest to sp
17:     S(is_c) ← the object segment producing p_c
18:     insert sub-segment (sp, p_c) into WP
19:     sp ← p_c
20:     n ← c
21: return WP
```

*Theorem 4:* 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.

**Proof**. The proof is similar to that of Theorem 2. □

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

### A. 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. 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 $S_{o_3}$ 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 $(p(S_{o_3}, ws), p(S_{o_3}, S_{o_2}))$. $S_{is_n}$ 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 $p(S_{o_2}, S_{o_3})$, 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 $(p(S_{o_1}, S_{o_2}), S^t_{o_1})$ is the working part for $S_{o_1}$.

### B. Why-Not Spatial Keyword Query Processing for Multiple Missing Objects

Following Theorem 4, 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 though 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 $\mathcal{D}$ 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 IV-B. 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 in the third step.

## VII. Empirical Study

### A. Experimental Setup

*1) Algorithms for Comparison:* We implemented four algorithms for study. The first is the baseline algorithm (**Baseline**, Section III-C), 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 [13] (**TM**). This last algorithm computes the rankings of the missing objects under all candidate weighting vectors. As stated by Theorem 2, 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.*, [8])—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 V to estimate the upper bound of the ranking under some weighting vector, *i.e.*, when $R(o, q, \vec{w_1})$ is being computed with another weighting vector $\vec{w_2}$, $R(o, q, \vec{w_1}) - R(o, q, \vec{w_2})$ will not exceed the number of promoted points between them.

*2) Datasets:* We use the real datasets EURO and GN in the experiments.[2] Both are commonly used in the spatial keyword query research [3], [4], [19], [26]. 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 IV.

TABLE IV. Dataset Characteristics

| | EURO | GN |
|---|---|---|
| total # objects | 162,033 | 1,868,821 |
| # distinct words | 35,315 | 222,407 |
| Avg. # words per object | 18 | 4 |

*3) 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.

[2]EURO — http://www.allstays.com; GN — http://geonames.usgs.gov

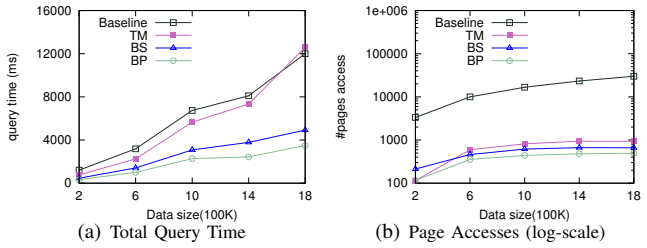
*4) Parameters:* We vary different system parameters. These parameters together with their default values (highlighted in bold) are shown in Table V. In the default setting, we select the object ranked $(10 \cdot k_0 + 1)st$ under $\vec{w_0}$ as the missing object.

TABLE V. PARAMETER SETTINGS

| Parameters | Settings |
|---|---|
| $k_0$ | 3,**10**,30,100,300 |
| # keywords | 2,**4**,8,16,32 |
| $\vec{w_0}$ | $<0.1, 0.9>$,$<0.3, 0.7>$, $<$**0.5**,**0.5**$>$,$<0.7, 0.3>$,$<0.9, 0.1>$ |
| $\lambda$ | 0.1, 0.3,**0.5**, 0.7, 0.9 |
| # missing objects | **1**,3,10,30 |

## B. Empirical Results

*1) 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. 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.



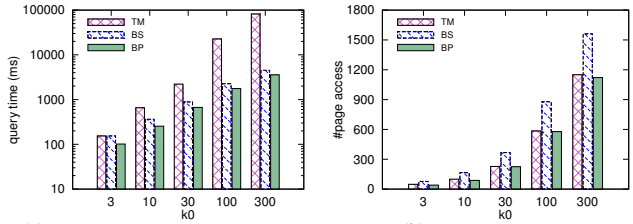
(a) Total Query Time (b) Page Accesses (log-scale)

Fig. 8. Varying Dataset Size

*2) 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. 9 plots the performance for the EURO dataset,[3] 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.

*3) 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. 10, 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

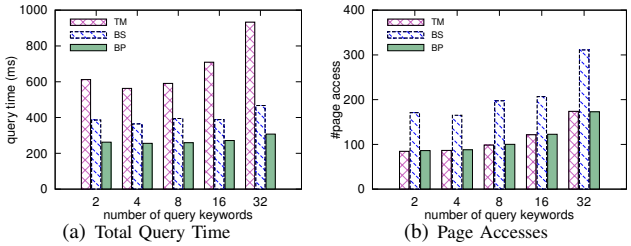
[3]In the interest of space, we omit the results for the GN dataset as they are similar.



(a) Total Query Time (log-scale) (b) Page Accesses

Fig. 9. Varying $k_0$

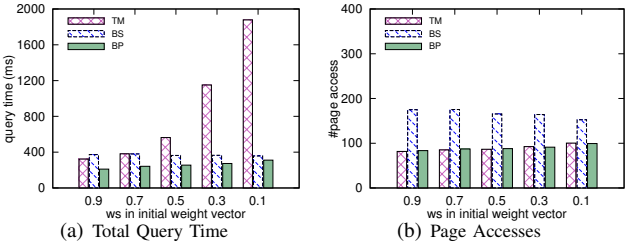
spatial keyword top-$k$ query many times, the computation of textual similarity becomes a dominating factor. This is why the query time of **TM** increases more noticeably with more query keywords.



(a) Total Query Time (b) Page Accesses

Fig. 10. Varying Number of Query Keywords

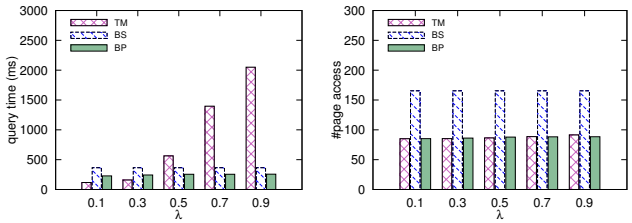
*4) Varying the initial weighting vector $\vec{w_0}$:* Next we investigate the effect of varying the weighting vector $\vec{w_0}$ in the initial spatial keyword top-$k$ query. As shown in Fig. 11, varying $\vec{w_0}$ has almost no impact on **BS** and **BP**. However, the query time of **TM** increases 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 $\vec{w_0}$.



(a) Total Query Time (b) Page Accesses

Fig. 11. Varying Initial Weighting Vector $\vec{w_0}$

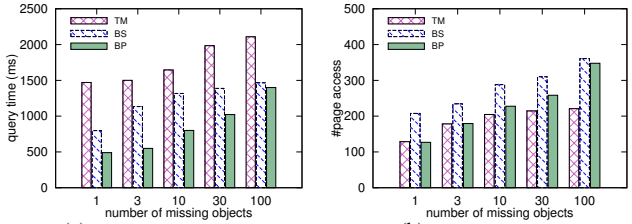
*5) Varying $\lambda$:* Parameter $\lambda$ allows the user to indicate the preference of modifying $k_0$ or the weighting vector $\vec{w_0}$. When varying $\lambda$, we observe from Fig. 12 that **BS** and **BP** are almost unaffected. The reason is that in **BS**, $\lambda$ is only used in the last 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. Thus, a larger $\lambda$ causes a larger initial penalty in **TM**. This further influences the optimizations of **TM** mentioned in Section VII-A1. Thus, **TM** is quite sensitive to changes in $\lambda$ and the query time increases dramatically with $\lambda$.




(a) Total Query Time (b) Page Accesses

Fig. 12. Varying $\lambda$

*6) 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. 13 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.




(a) Total Query Time (b) Page Accesses

Fig. 13. Varying Number of Missing Objects

## VIII. Conclusions and Future Work

In this paper, we studied how to answer 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.

So far we considered modifying the parameters $k$ and $\vec{w_0}$ when refining the original spatial keyword query. In future work, we plan to also consider the query keywords and the query location in why-not queries.

## Acknowledgements

This work is partially supported by HK-RGC GRF grants HKBU211512 & HKBU12202414 and HKBU FRG2/12-13/081. The work of Xin Lin was supported by China Postdoctoral Science Foundation, Shanghai Postdoctoral Scientific Program and Shanghai Pujiang Program. The authors thank Prof. Eric Lo for providing the source code of [13].

## References

[1] S. S. Bhowmick, A. Sun, and B. Q. Truong. Why Not, WINE?: Towards answering why-not questions in social image search. In *MM*, pp. 917-926, 2013.

[2] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. In *PVLDB*, pp. 1-12, 2012.

[3] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. In *PVLDB* 3(1): 373-384, 2010.

[4] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD* pp. 373-384, 2011.

[5] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. In *PVLDB*, pp. 217-228, 2013.

[6] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, pp. 277-288, 2006.

[7] A. Chapman and H. V. Jagadish. Why not?. In *SIGMOD*, pp. 523-534, 2009.

[8] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. In *PVLDB*, 2(1):337-348, 2009.

[9] S. Chester, A. Thomo, S. Venkatesh and S. Whitesides. Indexing for vector projections. In *DASFAA*, pp. 367-376, 2011.

[10] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pp. 656-665, 2008.

[11] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu. SEAL: Spatio-textual similarity search. In *PVLDB*, 5(9):824-835, 2012.

[12] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pp.47-57, 1984.

[13] Z. He and E. Lo. Answering why-not questions on top-k queries. In *ICDE*, pp. 750-761, 2012.

[14] M. Herschel and M. A. Hernández. Explaining missing answers to SPJUA queries. In *PVLDB*, 3(1-2):185-196, 2010.

[15] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. In *PVLDB*, 1(1):736-747, 2008.

[16] M. S. Islam, R. Zhou, and C. Liu. On answering why-not questions in reverse skyline queries. In *ICDE*, pp. 973-984, 2013.

[17] G. Li, J. Feng, and J. Xu. DESKS: Direction-aware spatial keyword search. In *ICDE*, pp. 474-485, 2012.

[18] F. Li, B. Yao, M. Tang, and M. Hadjieleftheriou. Spatial approximate string search. In *TKDE*, 25(6):1394-1409, 2012.

[19] J. Lu , Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pp. 349-360, 2010.

[20] C. Manning, P. Raghavan, and H. Schütze. Introduction to Information Retrieval. Cambridge University Press, 2008.

[21] B. Martins, M. J. Silva, and L. Andrade. Indexing and ranking in geo-IR systems. In *GIR*, pp. 31-34, 2005.

[22] J. B. Rocha-Junior and K. Nørvåg. Top-k spatial keyword queries on road networks. In *EDBT*, pp. 168-179, 2012.

[23] Y. Tao, X. Xiao, and J. Pei. Efficient skyline and top-k retrieval in subspaces. In *TKDE*, 19(8):1072-1088, 2007.

[24] Q. T. Tran and C. Chan. How to ConQueR why-not questions. In *SIGMOD*, pp. 15-26, 2010.

[25] D. Wu, Y. Li, B. Choi, and J. Xu. Social-aware top-k spatial keyword search. In *MDM*, 2014.

[26] D. Wu, M. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pp. 541-552, 2011.

[27] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pp. 688-699, 2009.

[28] D. Zhang, B. C. Ooi, and A. K. H. Tung. Locating mapped resources in web 2.0. In *ICDE*, pp. 521-532, 2010.

[29] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pp. 155-162, 2005.