Answering Why-Not Group Spatial Keyword Queries

Bolong Zheng, Kai Zheng, Christian S. Jensen, *Fellow, IEEE*, Nguyen Quoc Viet Hung, Han Su, Guohui Li, and Xiaofang Zhou, *Fellow, IEEE*

Abstract—With the proliferation of geo-textual objects on the web, extensive efforts have been devoted to improving the efficiency of top-k spatial keyword queries in different settings. However, comparatively much less work has been reported on enhancing the quality and usability of such queries. In this context, we propose means of enhancing the usability of a top-k group spatial keyword query, where a group of users aim to find k objects that contain given query keywords and are nearest to the users. Specifically, when users receive the result of such a query, they may find that one or more objects that they expect to be in the result are in fact missing, and they may wonder why. To address this situation, we develop a so-called *why-not* query that is able to minimally modify the original query into a query that returns the expected, but missing, objects, in addition to other objects. Specifically, we formalize the *why-not* query in relation to the top-k group spatial keyword query, called the Why-not Group Spatial Keyword Query (WGSK) that is able to provide a group of users with a more satisfactory query result. We propose a three-phase framework for efficiently computing the WGSK. The first phase substantially reduces the search space for the subsequent phases by retrieving a set of objects that may affect the ranking of the user-expected objects. The second phase provides an incremental sampling algorithm that generates candidate weightings of more promising queries. The third phase determines the penalty of each refined query and returns the query with minimal penalty, i.e., the minimally modified query. Extensive experiments with real and synthetic data offer evidence that the proposed solution excels over baselines with respect to both effectiveness and efficiency.

Index Terms—Spatial keyword queries, why-not, top-k query, query processing

1 Introduction

With the rapid deployment of location-based services and geo-positioning technologies, increasing amounts of geo-textual objects, or Point-of-Interests (PoIs), are available. A geo-textual object encompasses a geo-location and a textual description. There are now numerous online sources from which geo-textual objects can be acquired, including business directories such as Google My Business,[1] location-based social networks such as Foursquare,[2] as well as rating and review services such as TripAdvisor[3] and Dianping.[4] Making such objects conveniently available to users calls for techniques that offer efficient support for *spatial keyword queries* that take a location and a set of keywords as arguments and retrieve k objects that score the highest according to a ranking function that takes into account both spatial proximity and textual relevance [1], [2], [3], [4], [5], [6], [7].

Group Spatial Keyword Query. Most existing spatial keyword query techniques only support a single-user scenario. However, some decision making scenarios may involve multiple users. For example, several friends in a city may want to find a place to meet. A good meeting place may be one that minimizes the their overall travel, i.e., the sum of the distances they need to travel in order to meet. However, a number of other factors, such as transportation accessibility (walking, driving or taking subway), may also be taken into account. Furthermore, the relative tolerance to travel may be user-dependent; one user may be interested in minimizing the travel to reach a facility, while another user may be willing to accept longer travel if this reduces the monetary cost of the travel (toll fees, fuel consumption).

Unlike existing aggregate nearest neighbor queries that simply aggregate the distances from a data object to each query point [8], [9], [10], [11], [12], [13], this paper studies a more advanced and flexible query, called the top-k group spatial keyword query (GSK). Given a set of geo-textual objects D, a group of h users with different preferences, and a query keyword t_q, it finds the top-k objects from D containing t_q with the highest weighted sum scores. The score of an object o w.r.t. h users is computed as the sum of the h products of the user preferences and the spatial proximities. The preferences of the users form a weighting vector $\vec{w}$, where each preference indicates a user's travel tolerance.

1. https://www.google.com/business/
2. https://foursquare.com/
3. https://www.tripadvisor.com/
4. https://www.dianping.com

- *B. Zheng and G. Li are with the Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {bolongzheng, guohuili}@hust.edu.cn.*
- *K. Zheng and H. Su are with the Big Data Research Center, University of Electronic Science and Technology of China, Chengdu, Sichuan 611731, China. E-mail: {zhengkai, hansu}@uestc.edu.cn.*
- *C.S. Jensen is with the Aalborg University, Aalborg 9100, Denmark. E-mail: csj@cs.aau.dk.*
- *N.Q.V. Hung is with the Griffith University, Nathan, QLD 4111, Australia. E-mail: quocviethung1@gmail.com.*
- *X. Zhou is with the University of Queensland, St Lucia, QLD 4072, Australia. E-mail: zxf@itee.uq.edu.au.*

Manuscript received 1 Apr. 2018; revised 21 Oct. 2018; accepted 1 Nov. 2018. Date of publication 6 Nov. 2018; date of current version 5 Dec. 2019. (Corresponding author: Kai Zheng.) Recommended for acceptance by F. Silvestri. Digital Object Identifier no. 10.1109/TKDE.2018.2879819

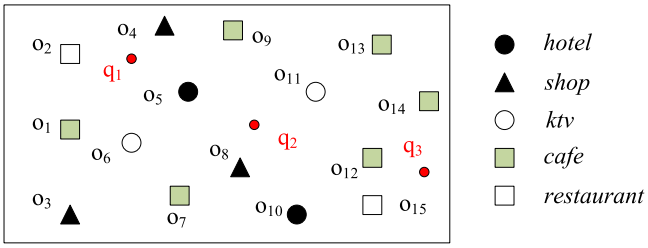


Fig. 1. Geo-textual objects.

A lower preference for a user means that the user has a lower tolerance and does not want to travel too far.

Motivation of the Why-not Query. While extensive efforts have been devoted to improving the efficiency of spatial keyword queries, such queries would also benefit from additional flexibility and expressiveness. For instance, users are required to specify their preferences along different dimensions (spatial, textual, or some other dimensions such as ratings and popularity); and in settings involving multiple users, it is even more difficult to quantify their preferences as a set of numeric weights. As a result, the quality and usability of such queries may fail to meet user expectations, e.g., certain user-desired objects are unexpectedly absent from query results. In order to improve the quality and usability of queries, the so-called *why-not* queries aim to explain why the expected results are not returned. Thus, why-not queries help users understand initial queries better and offer modified queries that contain the desired but previously missing objects in their results. The net effect is improvements in the usability of spatial keyword queries. Consider the example in Fig. 1:

Example 1. A group $\{q_1, q_2, q_3\}$ of users plan to find a cafe to meet. As they use different travel modes, their preferences form a weighting vector $(0.5, 0.3, 0.2)$. They issue a top-3 group spatial keyword query with the keyword "cafe". However, surprisingly, the Starbucks cafe (o_7) is not in the result $\{o_{12}, o_9, o_1\}$. The users wonder why the Starbucks cafe is not in the result. Is that because cafes with better locations have opened? Are the travel preferences not as expected? Can the Starbucks cafe be included in the result if the query is modified to a top-5 query instead of a top-3 query?

With this as motivation, we study the problem of answering the *why-not* question on top-k group spatial keyword queries, called Why-not Group Spatial Keyword Query (WGSK). In our setting, a group of users issue a top-k GSK query. However, objects expected to be in the result by one or more users do not appear in the result. A WGSK query is then able to provide an explanation of why the expected objects are missing, as well as provide a minimally revised query that includes the missing objects in its result.

Limitation of Existing Solutions. Several approaches have been proposed to answer *why-not* questions, including manipulation identification, database modification, and query refinement. The first category studies Select-Project-Join (SPJ) queries that aim to determine the manipulations that are responsible for excluding user-desired objects from a result [14], [15]. The second category focuses on providing database updates so that the missing objects appear in results [16], [17]. The third category revises initial query to generate a refined query whose result contains the missing objects. He et al. [18] adopt the third idea to top-k queries and study how to minimize the overall change of weights $\vec{w}$ and the parameter k while achieving the inclusion. However, their solutions only work for static datasets and do not apply to spatial keyword queries where query locations are dynamic and precomputation based on spatial distance is infeasible. Chen et al. [19] study how to answer *why-not* questions on top-k spatial keyword queries by also modifying $\vec{w}$ and k, but the changes on $\vec{w}$ are limited to spatial and textual dimensions, and they do not consider additional dimensions.

Contributions. We present a framework that provides a three-phase solution to answer why-not group spatial keyword queries. Using query refinement, we modify the users' preference vector $\vec{w}$ and the parameter k in the original top-k GSK query so that the expected answers are included in the result of the refined query. Based on a proposed penalty model, we present an efficient algorithm to generate promising refined queries by sampling weightings, and we find the one that modifies the original query minimally. In brief, the key contributions are summarized as follows:

- We formalize the why-not group spatial keyword query on top of the top-k group spatial keyword query. To our knowledge, there is no prior work on this problem.
- We propose a three-phase solution to process the why-not group spatial keyword query. The first phase substantially reduces the search space for the subsequent phases by efficiently retrieving a set of objects that may affect the ranking of user-expected objects. In the second phase, we propose an incremental sampling algorithm to generate candidate weightings. In the third phase, we determine the penalties of the refined queries and return the optimal one.
- We conduct an empirical study on real and synthetic PoI data. The study indicates that the paper's proposal is efficient and effective in terms of returning refined queries with the least penalty.

Roadmap. The remainder of the paper is organized as follows. We first formulate the Why-not Group Spatial Keyword Query (WGSK) in Section 2. Then we provide a solution overview in Section 3. Sections 4, 5 and 6 present the three phases of the solution, retrieving competitors, sampling weightings, and determining penalty, respectively. Section 7 reports on the experimental study, and Section 8 reviews related work. Finally, Section 9 concludes the paper.

2 Problem Statement

This section formalizes the setting and defines the top-k GSK query and the WGSK query. Frequently used notations are summarized in Table 1.

2.1 Setting

Definition 1 (Geo-textual Object). *Let D be a set of geo-textual objects, where each geo-textual object $o \in D$ has a location $o.l$ and a set of keywords $o.\phi$.*

Definition 2 (Utility Function). *Assume a group of h users' query locations $Q = \{q_1, \ldots, q_h\}$, and let $\vec{w} = (w_1, \ldots, w_h)$ be a weighting vector where each value w_i in $\vec{w}$ represents the ith user's preference (or tolerance) to distance. For each geo-textual*

TABLE 1
Summary of Notations

Notation	Definition
D	A dataset of geo-textual objects
$o(l, \phi)$	A geo-textual object o with location $o.l$ and a set of keywords $o.\phi$
$Q = \{q_1, \ldots, q_h\}$	A set of h users' query locations
$\vec{w} = (w_1, \ldots, w_h)$	A weighting vector of preferences
$d(q_i, o)$	The distance between q_i and o
$\vec{o}$	A vector representation of the object o
$u_{Q,\vec{w}}(\vec{o})$	The utility of object $\vec{o}$ under query Q and weighting vector $\vec{w}$
$M = \{\vec{m_1}, \ldots, \vec{m_n}\}$	The missing objects
$Penalty(k', \vec{w'})$	The penalty of a refined query $(k', \vec{w'})$
$rank(\vec{o}, \vec{w})$	The rank of $\vec{o}$ under $\vec{w}$
$H(\vec{o})$	The hyperplane between $\vec{o}$ and $\vec{m}$

object $o \in D$, the users' utility, $u_{Q,\vec{w}}(o)$, obtained from o is defined as follows:

$$u_{Q,\vec{w}}(o) = \sum_{i=1}^{h} w_i \cdot (1 - d(q_i, o)), \tag{1}$$

where $d(q_i, o)$ is a function that normalizes the euclidean distance between q_i and o into the range $[0, 1]$. When the context is clear, we simply use $u_{\vec{w}}(o)$ instead of $u_{Q,\vec{w}}(o)$.

Without loss of generality, we assume that $0 < w_i < 1$ for $i \in [1, h]$, and that $\sum_{i=1}^{h} w_i = 1$. The intuition behind the utility function includes two aspects:

(i) the spatial proximity is defined differently for different users, depending on their means of movement. Due to different transportation modes, the tolerance (sensitivity) towards distance is user-dependent, and varies from user to user;
(ii) summing up the weighted spatial proximities indicates an overall degree of benefit to a group of users of an object. Moreover, this model is widely adopted in existing studies of top-k preference queries [20], [21], [22], [23].

Note that the normalization of the relative tolerances in $\vec{w}$ does not restrain the semantics of the utility function [24].

Object Vectorization. It is worth noting that once Q is specified, the distances between all $q_i \in Q$ and each object $o \in G$ are constant values. Therefore, for each o, we have an h-dimensional vector $\vec{o} = (1 - d(q_1, o), \ldots, 1 - d(q_h, o))$, where for $i \in [1, h]$, $\vec{o}[i]$ denotes the spatial proximity between o and $q_i \in Q$. In the following, we use $\vec{o}$ and object o interchangeably when this does not cause ambiguity. Hence Equation (1) can be rewritten as

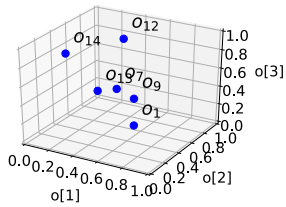
$$u_{\vec{w}}(\vec{o}) = \vec{o}^{\,\intercal} \cdot \vec{w}. \tag{2}$$

Example 2. In Fig. 2, let $Q = \{q_1, q_2, q_3\}$ be given and consider object o_1: The distances between o_1 and $\{q_1, q_2, q_3\}$ are normalizes to $(0.4, 0.5, 0.9)$. Thus o_1 is represented as $\vec{o_1} = (0.6, 0.5, 0.1)$. Given a weighting vector $\vec{w} = (0.5, 0.3, 0.2)$, the utility of $\vec{o_1}$ is 0.47.

2.2 Problem Definition

Definition 3 (Group Spatial Keyword Query). *Given h query locations Q, a query keyword t_q, a weighting vector $\vec{w}$, and an integer k, the top-k group spatial keyword query (GSK) aims to find a set S of up to k objects from D that all contain keyword t_q and have the highest utilities, i.e.,*

$$S = \{o \in D \,|\, t_q \in o.\phi, \forall o \in S, o' \in D \setminus S, u_{\vec{w}}(\vec{o}) > u_{\vec{w}}(\vec{o'})\}. \tag{3}$$



(a) Preference space

$1 - d(q_i, o_i)$	o_1	o_7	o_9	o_{12}	o_{13}	o_{14}
q_1	0.6	0.4	0.6	0.4	0.3	0.1
q_2	0.5	0.6	0.5	0.7	0.5	0.4
q_3	0.1	0.4	0.4	0.9	0.4	0.8

(b) Spatial proximity

Fig. 2. Objects in preference space.

Example 3. Building on Example 1, we have $Q = \{q_1, q_2, q_3\}$, a query keyword $t_q =$ "cafe", and a weighting vector $\vec{w} = (0.5, 0.3, 0.2)$. Therefore, the top-3 GSK query returns $S = \{o_{12}, o_9, o_1\}$.

After a GSK query $(Q, t_q, k_o, \vec{w_o})$ is issued and the result set S is obtained, the users may find that one or more objects expected to be in the result, i.e., $M = \{\vec{m_1}, \ldots, \vec{m_n}\}$, do not appear in S. We assume that all $\vec{m_i} \in M$ contain the query keyword t_q. Now, the user may pose a *why-not* query in order to obtain a refined GSK query $(k_b, \vec{w_b})$, where the original $\vec{w_o}$ and k_o are modified and the objects in M are included in the result of the query, S_b. Note that we use $(k, \vec{w})$ as an abbreviation for the GSK query $(Q, t_q, k, \vec{w})$ because we do not alter Q and t_q. Basically, we want to return a modified query that is as similar as possible to the original query. Therefore, we introduce a penalty model that quantifies the difference between an original query and a modified query.

Penalty Model. Let us assume that only one object $\vec{m}$ is missing. When modifying the original GSK to bring $\vec{m}$ back, we use Δk and Δw to measure the quality of the refined query, where $\Delta k = \max(0, k' - k_o)$ and $\Delta w = ||\vec{w'} - \vec{w_o}||_2$, and we let r_o denote the rank of $\vec{m}$ under original $(k_o, \vec{w_o})$. By adapting an existing penalty model [18], [19], the penalty of a refined query $(k', \vec{w'})$ is defined as follows:

$$Penalty(k', \vec{w'}) = \lambda \cdot \frac{\Delta k}{r_o - k_o} + (1 - \lambda) \cdot \frac{\Delta w}{\sqrt{1 + \sum w_o[i]^2}}, \tag{4}$$

where $\lambda \in (0, 1)$ is a user-specified parameter. For ease of use, we provide 5 settings for λ for users to choose among, as shown in Fig. 3. We know Δk is no larger than, and is normalized by $r_o - k_o$, and Δw is normalized to the unit range by $\sqrt{1 + \sum w_o[i]^2}$. Thus, both Δk and Δw are normalized. Intuitively, the lower the penalty is, the more satisfied the users are with the modified query.

Definition 4 (Why-not Group Spatial Keyword Query). *Given a top-k GSK query $(Q, t_q, \vec{w_o}, k_o)$, and a set of missing objects $M = \{\vec{m_1}, \ldots, \vec{m_n}\}$, the why-not group spatial keyword query (WGSK) returns a refined query $(k_b, \vec{w_b})$ with the lowest modification penalty and whose result includes all objects in M.*

We assume that the group of users includes more than two users, since the case of two users can be handled easily by modifying an existing method [19].

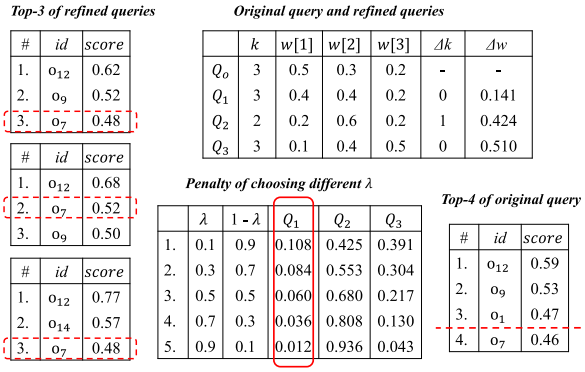


Fig. 3. Example of why-not GSK queries.

Example 4. We build on Example 1 and consider a group of users $\{q_1, q_2, q_3\}$ who plan to find a cafe shop where they can meet. The original query then has parameters t_q ="cafe", $k_o = 3$, and $\vec{w}_o = (0.5, 0.3, 0.2)$. The users expect object o_7 is in the result, but the object is missing. Fig. 3 shows that object o_7 is ranked 4th and is missing from the result, $\{o_{12}, o_9, o_1\}$. A why-not query is issued to bring o_7 back. Three refined queries are available for the users to choose among, i.e., $(k_1 = 3, \vec{w}_1 = (0.4, 0.4, 0.2))$, $(k_2 = 2, \vec{w}_2 = (0.2, 0.6, 0.2))$, and $(k_3 = 3, \vec{w}_1 = (0.1, 0.4, 0.5))$. We can see $(k_1, \vec{w}_1)$ ranks o_7 3-rd, $(k_2, \vec{w}_2)$ ranks o_7 2-nd, and $(k_3, \vec{w}_3)$ ranks o_7 3-rd. The corresponding Δk and Δw are also shown in Fig. 3. Given the 5 settings of λ, we can see $(k_1, \vec{w}_1)$ is the refined query with minimum penalty in all 5 cases.

3 Problem Analysis and Solution Overview

In this section, we analyse the problem and provide a solution overview for computing the WGSK query. For simplicity, we consider only one missing object $\vec{m}$.

3.1 Dominance Relationship

When a WGSK query $(Q, t_q, k_o, \vec{w}_o)$ with missing object $\vec{m}$ is issued, all objects that contain keyword t_q are easily obtained. The closer an object is to the users' locations, the more desirable it is. Next we define the dominance relationship between two objects.

Definition 5 (Dominance). *Given objects $\vec{a}$ and $\vec{b}$, if $\vec{a}[i] \leq \vec{b}[i]$ for $i \in [1, h]$ and at least one dimension exists where $\vec{a}[i] < \vec{b}[i]$, $\vec{a}$ is dominated by $\vec{b}$, denoted as $\vec{a} \prec \vec{b}$. Otherwise, $\vec{a} \not\prec \vec{b}$.*

For example, in Fig. 2, the spatial proximity at each dimension of $\vec{o_{12}}$ exceeds that of $\vec{o_{14}}$, meaning that $\vec{o_{14}} \prec \vec{o_{12}}$.

Observation 1. If an object $\vec{o}$ dominates the missing object $\vec{m}$, the utility of $\vec{o}$ exceeds that of $\vec{m}$, i.e., $u_{\vec{w}}(\vec{o}) > u_{\vec{w}}(\vec{m})$ for any weighting vector [25]. In other words, no matter how we choose the weighting vector, $\vec{o}$ is always more preferable than $\vec{m}$. Likewise, if $\vec{o} \prec \vec{m}$, we have $u_{\vec{w}}(m) > u_{\vec{w}}(o)$.

Based on this observation, we have the following definition of *competitor*:

Definition 6 (Competitor). *Given a missing object $\vec{m}$ and an object $\vec{o}$ that also contains query keyword t_q, if $\vec{m}$ does not dominate $\vec{o}$, and vice versa, we say that $\vec{o}$ is a competitor w.r.t. $\vec{m}$.*

Categories of Candidate Objects. Based on the notations of *dominance* and *competitor*, the objects that contain t_q can be partitioned into categories I_1, C, and I_2. Sets I_1 and I_2 contain *non-competitors* such that each object in I_1 dominates $\vec{m}$ and each object in I_2 is dominated by $\vec{m}$. Set C contains *competitors* w.r.t. $\vec{m}$.

Example 5. If o_7 is the missing object in Fig. 2 then objects $\{o_1, o_9, o_{14}\}$ are its competitors. Moreover, we have $I_1 = \{o_{12}\}$ and $I_2 = \{o_{13}\}$.

The Rank of Competitors. It is easy to see that any object in I_1 and I_2 always scores higher or lower than $\vec{m}$. The higher an object scores, the higher it ranks among all objects. Thus, we define the rank of an object $\vec{o}$ as follows:

$$rank(\vec{o}, \vec{w}) = 1 + |\{\vec{o'} : u_{\vec{w}}(\vec{o}) < u_{\vec{w}}(\vec{o'}), t_q \in o'.\phi\}|.$$

It suffices to know that the relative ranks of objects in I_1 or I_2 w.r.t. $\vec{m}$ remain unchanged and that these objects do not affect WGSK query processing. Therefore, only the *competitors* are taken into consideration in the later phases of query processing.

3.2 Hardness of the Problem

Answering a WGSK query amounts to modifying k_o and $\vec{w}_o$ in the original query to increase the rank of $\vec{m}$ while keeping the penalty as low as possible. A naive approach would be to increase k_o to $\vec{m}$'s original rank under $\vec{w}_o$. However, this solution fails to deliver the desired result. Rather, we need to find a refined query with new values for both k and $\vec{w}$ in order to minimize the penalty.

Preparation for Increasing the Rank. To increase the rank of a missing object, the first step is to extract the competitors that may affect its rank. This extraction of objects from the dataset may benefit from the availability of a spatio-textual index that enables filtering according to both the textual information and spatial dominance relationships. Suppose we have already obtained the set C of competitors with cardinality c. Then the rank of $\vec{m}$ is in the range $[|I_1| + 1, |I_1| + c + 1]$ depending on $\vec{w}$. To keep the presentation simple, we assume that the objects in I_1 and I_2 are removed from D, so we have $rank(\vec{m}) \in [1, c + 1]$.

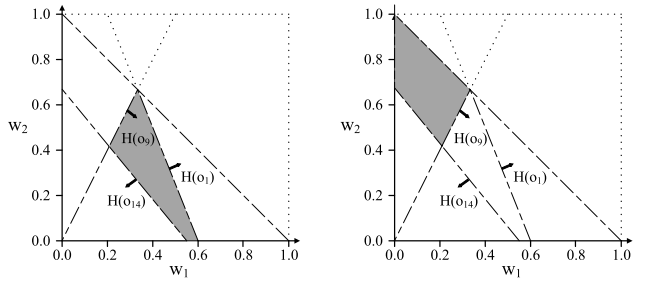
Partition Hyperplane. Given an object $\vec{o} \in C$ and the missing object $\vec{m}$, the equation $u_{\vec{w}}(\vec{o}) = u_{\vec{w}}(\vec{m})$ corresponds to a partitioning hyperplane $H(\vec{o})$ in the preference space, such that every weighting vector that falls on this hyperplane renders $\vec{o}$ and $\vec{m}$ equally preferable.

Observation 2. Hyperplane $H(\vec{o})$ partitions the preference space into two halfspaces, $H^+(\vec{o})$ and $H^-(\vec{o})$. For every $\vec{w}$ in $H^+(\vec{o})$, we have $u_{\vec{w}}(\vec{o}) > u_{\vec{w}}(\vec{m})$. Likewise, in $H^- u_{\vec{w}}(\vec{o})$, we have $u_{\vec{w}}(\vec{o}) < u_{\vec{w}}(\vec{m})$.

Given that the rank of $\vec{m}$ is $j + 1$, $j \in [0, c]$, j competitors exist that have scores higher than $\vec{m}$, and $c - j$ competitors exist that have scores lower than $\vec{m}$. With the constraints $w_i \in (0, 1)$, and $\sum_{i=1}^{h} w_i = 1$, we obtain the following inequalities:

$$\begin{cases} H^+(\vec{o}_{r_1}) : u_{\vec{w}}(\vec{o}_{r_1}) = \vec{o}_{r_1} \cdot \vec{w} > u_{\vec{w}}(\vec{m}) = \vec{m} \cdot \vec{w} \\ \cdots\cdots \\ H^+(\vec{o}_{r_j}) : u_{\vec{w}}(\vec{o}_{r_j}) = \vec{o}_{r_j} \cdot \vec{w} > u_{\vec{w}}(\vec{m}) = \vec{m} \cdot \vec{w} \\ H^-(\vec{o}_{r_{j+2}}) : u_{\vec{w}}(\vec{o}_{r_{j+2}}) = \vec{o}_{r_{j+2}} \cdot \vec{w} < u_{\vec{w}}(\vec{m}) = \vec{m} \cdot \vec{w} \\ \cdots\cdots \\ H^-(\vec{o}_{r_{c+1}}) : u_{\vec{w}}(\vec{o}_{r_{c+1}}) = \vec{o}_{r_{c+1}} \cdot \vec{w} < u_{\vec{w}}(\vec{m}) = \vec{m} \cdot \vec{w} \\ \forall i \in [1, h], w_i \in (0, 1), \sum_{i=1}^{h} w_i = 1 \end{cases} \quad (5)$$

Here, $\vec{o}_{r_i}$ is the competitor with rank i.



(a) $\{H^-(o_1), H^+(o_9), H^-(o_{14})\}$ (b) $\{H^-(o_1), H^-(o_9), H^-(o_{14})\}$

Fig. 4. Convex polytopes for missing object o_7.

Solving by Quadratic Programming. The solution to this system of inequalities is an infinite set of weighting vectors that forms a convex polytope and every $\vec{w}$ in it ranks $\vec{m}$ as $(j+1)$-st.

Example 6. In the example in Fig. 4, arrows indicate positive halfspaces partitioned by hyperplanes. Therefore, the polytope in Fig. 4a is constructed by $\{H^-(o_1), H^+(o_9), H^-(o_{14})\}$, and we have $rank(o_7) = 2$. Likewise, in Fig. 4b, the polytope is bounded by $\{H^-(o_1), H^-(o_9), H^-(o_{14})\}$, and $rank(o_7) = 1$.

For each such convex polytope, we need to find the $\vec{w}$ such that $\Delta w = ||\vec{w} - \vec{w_o}||_2$ is minimized, which can be solved by applying a quadratic programming solver [26].

Theorem 1. *The time complexity of exactly computing* WGSK *is* $O(\mathsf{cost}_1 + 2^c \cdot \mathsf{cost}_{qp})$.

Proof. We first assume that the cost of extracting competitors is $O(\mathsf{cost}_1)$. For each rank $j+1$ of $\vec{m}$, j competitors exist that score higher than $\vec{m}$. These form C_c^j combinations with corresponding convex polytopes. Therefore, for all possible rankings, we have $\sum_{j=0}^{c} C_c^j = 2^c$ convex polytopes in the worst case. We assume that the cost of answering an instance of Equation (5) with a solver is $O(\mathsf{cost}_{qp})$. Therefore, the time complexity of WGSK is $O(\mathsf{cost}_1 + 2^c \cdot \mathsf{cost}_{qp})$, which is impractical given a large number of competitors. □

3.3 Solution Overview

As described above, it is expensive to obtain an optimal refined query. Instead of computing an exact result, we target a good approximate solution by generating a set of weighting vectors that are near optimal. Intuitively, the more weighting vectors we have, the higher the chance we have of finding a good approximation. In addition, after generating a set of weighting vectors, we also need to consider the cost of determining the penalty of each refined query. To this end, we design a sampling algorithm to enable a tradeoff between the approximation quality and the computational cost. Moreover, a well-designed algorithm for penalty computation is desirable.

Fig. 5 illustrates the three steps of our framework that are also described briefly below; the algorithmic details of the framework are presented in Sections 4, 5, and 6.

(i) *Retrieving competitors*. This step finds a set of competitors C as the initial candidates. In this step, we build an IR^2-tree [27] on D. An IR^2-tree integrates signature files into the nodes of an R-tree, such that each node contains two types of information: (i) the MBR of its subtree and (ii) a signature file, which is the union of all signatures of its entries. We propose a geo-textual dominance search algorithm on top of the IR^2-tree to extract the competitors from D.

(ii) *Sampling weightings*. This step samples a set of candidate weighting vectors, which is a critical step in our solution that must take into account both the approximation quality and the computation efficiency. Specifically, we propose an incremental sampling approach that considers three different heuristic strategies against a random sampling approach.

(iii) *Determining penalty*. After the weighting vectors are generated, this step aims to find the weighting vector with lowest penalty. Instead of computing the penalty of all weighing vectors, we propose a branch-and-bound algorithm that disregards vectors with costs that are not competitive.

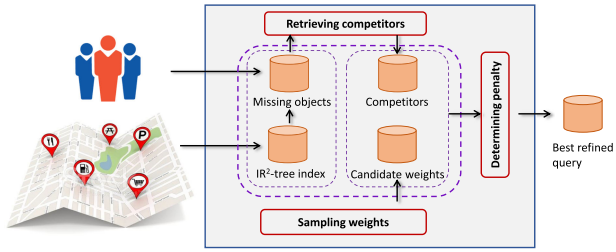


Fig. 5. Solution overview.

4 Retrieving Competitors

We proceed to present a geo-textual dominance search (GTD) algorithm that retrieves the competitors of $\vec{m}$ from D. It is straightforward to first adopt a keyword Boolean filtering that selects the geo-textual objects whose textual descriptions contain t_q. Then, for each of such object $\vec{o}$, we can compare it with $\vec{m}$ to determine whether it is a competitor. This approach can be supported using simply an inverted index. However, database D may contain a large amount of objects, especially for frequent keywords. Therefore, this approach, which may need to perform a linear scan of D in the worst case, is unlikely to perform well. The main idea in GTD is to follow the divide-and-conquer paradigm to search an IR^2-tree on D to retrieve competitor set C efficiently. GTD considers both the textual information and spatial dominance for the object filtering. In addition to utilizing the IR^2-tree to find competitors, we can also use it to answer the GSK query and computing r_o for $\vec{m}$ in the original query, which serves as a prerequisite for the *why-not* question. We disregard the details since this is not the focus of the paper.

4.1 Search Algorithm

The GTD algorithm first initializes a set C that will contain competitors and an empty heap H to hold entries (either nodes or objects) from the IR^2-tree. Let e be an entry with MBR M_e. Let $MINDIST(q_i, M_e)$ be the minimum and $MAXDIST(q_i, M_e)$ be the maximum normalized distance between point q_i and MBR M_e. For the processing of entries, we have the following:

Lemma 1. *For an entry e and the missing object $\vec{m}$, if for all dimensions $i \in [1, h]$, we have*

$$m[i] \geq 1 - MINDIST(q_i, M_e),$$

then all objects in M_e are dominated by $\vec{m}$ and belong to I_2. Likewise, if for all i, we have

$$m[i] \leq 1 - MAXDIST(q_i, M_e),$$

then all objects in M_e dominate $\vec{m}$ and belong to I_1.

Algorithm 1. Geo-Textual Dominance Search (GTD)

Input: $\{Q, t_q, \vec{w}_o, k_o, m\}$ and IR^2-tree on D
Output: The set of competitors C

```
1: Initialize an empty set C, an empty heap H;
2: Add root node into H;
3: while H is not empty do
4:   e ← top entry of H;
5:   if e is non-leaf entry then
6:     for each child e' of e do
7:       check if M'_e dominates or is dominated by m⃗;
8:       if yes, we skip; otherwise, insert e' into H;
9:   else
10:    for each o in e do
11:      check if o is a competitor of m⃗ and insert into C if yes;
12:  return C;
```

Algorithm Outline. Initially, we insert the root entry into H. Then GTD iteratively removes the top entry e from H and then performs the following operations depending on the entry type, and it terminates when H becomes empty.
Checking a Non-Leaf Node. We consider a combination of a textual and spatial check for the pruning. Given a node e, we proceed as follows.

(i) We first check $s(e') \wedge s(t_q)$ for each of its child entries e'. If $s(e') \wedge s(t_q)$ is zero, e' is discarded immediately. Otherwise, we continue to perform a dominance check to see if e' is dominated by $\vec{m}$ or dominates $\vec{m}$;
(ii) To determine which category the entry e' belongs to, we use a flag to distinguish it based on Lemma 1. For $i = 1$, we first compare $m[1]$ with $1 - MINDIST(q_1, M_e)$. If $m[1] > 1 - MINDIST(q_i, M_e)$, we set the current flag to I_2. Otherwise, we set it to I_1. Then we proceed to check other dimensions. If the flag of a new dimension violates the current flag then the entry is likely to contain competitors and is inserted into heap H. It is worth noting that we do not have to finish the remaining dimensions, since e' now has no chance of being dominated by or to dominate $\vec{m}$. If all dimensions are processed and no flag violations occur, then e' is dismissed since all the objects covered by e' are either in I_1 or in I_2.

Checking a Leaf Node. If e is a leaf node then it contains objects only. Thus, we conduct a same keyword check. For each object $\vec{o}$, if $s(o) \wedge s(t_q)$ is zero, it cannot be considered as a competitor. Otherwise, GTD performs the same dominance check. The difference is that we compare $m[i]$ with $o[i]$ directly. If $\vec{o}$ is a competitor, we insert $\vec{o}$ into C.

Theorem 2 (Correctness of GTD). *The* GTD *algorithm correctly reports all competitors in C.*

Proof. We prove this by contradiction. Assume $\vec{o}$ is a competitor but is pruned by GTD. It is easy to see that $\vec{o}$ will not be pruned in a leaf node. For a non-leaf node e that contains $\vec{o}$, if e is pruned then we know $\forall \vec{o'} \in e$, it holds for $i \in [1, h]$ that $m[i] \geq o'[i]$ or $m[i] \leq o'[i]$, which contradicts the assumption. Thus, the theorem follows. □

5 Sampling Weightings

We proceed to describe our method for sampling weights $\vec{w}$ from the preference space. As previously discussed, it is impractical to find the optimal refined query in the infinite preference space, but we can find a good approximation of the optimal $\vec{w}$ by means of a trade-off between the quality of the refined query and the running time. The basic idea of sampling weights is as follows. First, we sample a certain number of weighting vectors $W = \{\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_s\}$. Then for each $\vec{w}_i \in W$, we compute the corresponding k_i that introduces $\vec{m}$ into the result with minimal penalty. Specifically, we take the following issues into consideration: (i) Where to obtain a set of weighting vectors with high quality and (ii) how to choose an appropriate number of weighting vectors. Next, we introduce a random sampling approach (RSA) [18] as a baseline algorithm, and then we present our incremental sampling approach (ISA).

5.1 Random Sampling Approach

Recall that the rank of $\vec{m}$ belongs to $[1, c+1]$. For each possible rank i, an infinite set of weighting vectors W_{r_i} exist that are built by C_c^i convex polytopes. Therefore, a set $\mathcal{W} = \cup_{r_i=1}^{c+1} W_{r_i}$ exists from which we can sample a finite set of weighting vectors for processing the next step. If we sample a $\vec{w}$ in W_{r_i} then a refined query $q(\vec{w}, r_i)$ is obtained. Then the penalty can be determined after we compute the rank r_i of $\vec{m}$ under the refined $\vec{w}$.

5.1.1 Sampling Space

We first introduce an important theorem that indicates where we can obtain weighting vectors that are good approximations of the optimal one.

Theorem 3. *If the original query $q_o(\vec{w}_o, k_o)$ is not the optimal answer then the optimal refined query q_{opt} with the minimum penalty has a weighting vector $\vec{w}_{opt}$ on the boundary of the union weighting vector set $\mathcal{W}$.*

Proof. As we know, each inequality system given in Equation (5) corresponds to a convex polytope. The boundary of the convex polytope is exactly the hyperplane where competitors and missing objects have the same score. Therefore, the weighting vectors on the boundary rank $\vec{m}$ higher or equal to those inside the convex polytope. In addition, the weighting vectors on the boundary have a smaller Δw. Therefore, the weighing vector $\vec{w}_{opt}$ of the optimal refined query q_{opt} must be on the boundary of $\mathcal{W}$. Interested readers may refer to [18] for a detailed proof. □

Theorem 3 tells that in order to obtain a good approximation, we can sample candidate weighting vectors on the boundary of $\mathcal{W}$. Instead of computing the boundary of each convex polytope, which is too time consuming, we can directly use the partition hyperplane of each competitor $\vec{o}$ given by $\vec{w} \cdot (\vec{m} - \vec{o}) = 0$ in Equation (6), which collectively forms the boundary of the convex polytopes.

$$\begin{cases} H(\vec{o}) : u_{\vec{w}}(\vec{o}) = \vec{o} \cdot \vec{w} = u_{\vec{w}}(\vec{m}) = \vec{m} \cdot \vec{w} \\ \forall i \in [1, h], w_i \in (0, 1), \sum_{i=1}^{h} w_i = 1. \end{cases} \quad (6)$$

5.1.2 Sampling Cardinality

Intuitively, the more weighting vectors we sample, the more likely we are to find one that is close to the optimal one. However, considering more weighting vectors also increase the running time, since for each weighting vector $\vec{w}$, we have to determine the exact rank of $\vec{m}$ under $\vec{w}$ in the penalty determination step. We employ a general equation below to decide on a cardinality of the weighting vector set

$$s \geq \log(1 - Pr)/\log(1 - T\%). \tag{7}$$

Specifically, if we hope the probability of getting at least one of the best $T\%$ weighting vectors to be sampled is no smaller than Pr, the cardinality of samples s must be larger than a certain threshold. From Equation (7) we know the cardinality of the weighting vector set is independent of the cardinality of the set of competitors. Therefore, an appropriate setting of $T\%$ and Pr is required to control the quality of the samples. Intuitively, the quality can be improved by generating more samples if we decrease $T\%$ and enlarge Pr, which is verified by experiments in Section 7.

5.2 Incremental Sampling Approach

Although the random sampling approach may likely generate promising weighting vectors, the quality, however, cannot be guaranteed. The random sampling approach suffers from two shortcomings:

(i) The weighting vectors are sampled randomly on different partition hyperplanes, so it is possible that many vectors are taken from less promising hyperplanes, which reduces the possibility of considering the optimal weighting vectors;
(ii) If many weighting vectors are taken from unpromising hyperplanes, computational resources are wasted on penalty computations.

This motivates an incremental sampling approach (ISA) that devotes particular attention to selecting high quality weighting vectors. In order to overcome these shortcomings, we take all hyperplanes into consideration by first sampling a weighting vector from each hyperplane, then continue to generate the rest samples from the more promising ones. Therefore, we are supposed to generate at least c weighting vectors to make ISA work. Fortunately, we always have s larger than c determined by an appropriate setting of $T\%$ and Pr for an acceptable sampling quality. In the experimental setting of Section 7, all the evaluated queries fall into this category.

Algorithm Initialization. We first initialize a priority queue PQ. For each competitor $\vec{o}$, we sample a weighting vector from the partition hyperplane $H(\vec{o})$ between $\vec{o}$ and $\vec{m}$. Then we insert each hyperplane $H(\vec{o})$ into PQ based on the score of the weighting vector sampled from it.

Updating Priority Queue. As s exceeds c, we have $s - c$ weighting vectors left to sample. To do that, we invoke a procedure SelectHyper() that selects the hyperplane from which to sample the next candidate weighting vector. In each round, we pop the hyperplane with the highest score from PQ and sample a weighting vector on it. Then we recompute the score of the hyperplane based on particular heuristics adopted by SelectHyper() and insert it into PQ again. The intuition is that we want to sample the next weighting vector from the currently most promising hyperplane. This process terminates when all s samples are obtained. Algorithm 2 describes the details of the incremental sampling approach. In SelectHyper(), we use three selection strategies based on different heuristics.

Algorithm 2. Incremental Sampling Approach(ISA)

Input: The competitors C, $\vec{m}$, T, and Pr
Output: The set of weighting vectors W

```
1: s ← log(1 − Pr)/log(1 − T%);
2: Initialize a priority queue PQ of size c;
3: for each H(o) do
4:    Sample a w and insert to W;
5:    Compute a score based on heuristics;
6:    Insert H(o) into PQ;
7: while c < s do
8:    Pop H from PQ;
9:    Sample a w on H and insert it into W;
10:   Compute a score based on heuristics;
11:   Insert H into PQ;
12:   c++;
13: return W;
```

Utility Score (US) Based Strategy. The straightforward approach to selecting the next hyperplane to sample from is to pick the one that scores the missing object with the highest utility among all c hyperplanes. The reason is that the hyperplane with the highest utility score is likely to rank the missing object the highest. In detail, we sample a weighting vector $\vec{w}$ from each $H(\vec{o})$, compute $u_{\vec{w}}(\vec{m})$, and insert $H(\vec{o})$ into PQ according to $u_{\vec{w}}(\vec{m})$. Then SelectHyper() selects $H(\vec{o})$ the top element from PQ and resamples a $\vec{w'}$. After computing $u_{\vec{w'}}(\vec{m})$, we insert $H(\vec{o})$ into PQ again.

Weight Modification (WM) Strategy. The US strategy suffers from the fact that the increase in the utility score does not directly reflect the extent of the modification of the original query. Even if the utility score of a sampled weighting vector is high, the modification of the original query is uncertain, and it is possible that the penalty of a refined query is not competitive. So we consider a supervised method where we continue to sample the next $\vec{w'}$ on the hyperplane $H(\vec{o})$ with the current minimum Δw. The rest of the algorithm follows the US strategy.

Rank Improving (RI) Strategy. The WM strategy aims to minimize the modification of weight by sampling from the hyperplane with the sampled weighting vector that maintains the current minimum Δw, but does not guarantee the modification of the rank of $\vec{m}$. Therefore, we take a step further to consider both weight modification and rank improvement. For each hyperplane $H(\vec{o})$, we sample a weighting vector $\vec{w}$ and randomly choose a constant number η of competitors from C. Then we compute the utility of each competitor under $\vec{w}$ and compare it with that of $\vec{m}$. Assuming that τ competitors score higher than $\vec{m}$, from the rank of $\vec{m}$ among the subset of competitors, we can roughly estimate $rank(\vec{m}, \vec{w})$ as follows:

$$\widetilde{rank(\vec{m}, \vec{w})} = \left\lfloor \frac{\tau}{\eta} \cdot c \right\rfloor + 1. \tag{8}$$

Then we can use Δw and $\widetilde{rank(\vec{m}, \vec{w})}$ to estimate the penalty and insert the corresponding hyperplane into PQ.

Time Complexity. Assume the time for sampling a weighting vector is $O(f)$. The random sampling approach (RSA) sequentially samples s weighting vector, so the time for RSA is $O(s \cdot f)$. The incremental sampling approach (ISA)

needs to sample the same number of weighting vectors, but it maintains a PQ with size c and requires additional costs to insert a new element into PQ. Therefore, the time complexity of ISA is $O(s \log c \cdot f)$.

6 Determining Penalty

We proceed to present the algorithm for determining penalties of refined queries. Given a weighting vector $\vec{w'}$, we need to compute $rank(\vec{m}, \vec{w'})$ and form the refined query $(k', \vec{w'})$, where $k' = rank(\vec{m}, \vec{w'})$. Then we compare the penalty of all refined queries and return the one with minimum penalty. We notice that the penalty determination roughly corresponds to the rank-aware processing that occurs in the reverse top-k query. Given an object $\vec{m}$ and a set of weighting vectors, the reverse top-k query identifies all weighting vectors for which $\vec{m}$ belongs to the top-k result set. The main differences between penalty validation and reverse top-k query are that: (i) we need to exactly determine $rank(\vec{m}, w')$ of $\vec{m}$, not just to check if $\vec{m}$ is one of the top-k results; and (ii) only the weighting vector with minimum penalty will be reported, not all weighting vectors that include $\vec{m}$ as a top-k result.

It is easy to see that the brute force approach to computing $rank(\vec{m}, \vec{w'})$ under each $\vec{w'}$ requires $O(s \cdot c)$ time for s weighting vectors and c competitors, since $rank(\vec{m}, \vec{w'})$ is simply the number of objects score higher than $\vec{m}$. To avoid redundant computations, a progressive top-k algorithm can be applied to speed up the computation of $rank(\vec{m}, \vec{w'})$ for each weighting vector $\vec{w'}$ and to prune unnecessary weighting vectors. However, the progressive top-k algorithm suffers from two main drawbacks: (i) it requires access to all weighting vectors sampled in the previous step; and (ii) it cannot avoid executing the top-k query for each of them. Therefore, we instead introduce a branch-and-bound algorithm for efficient penalty determination.

6.1 Progressive Top-k Algorithm

To apply a progressive top-k algorithm, we can directly adopt any existing method, such as [23], to determine $rank(\vec{m}, \vec{w})$. The details of the algorithm are omitted, but two pruning conditions are discussed.

Upper Bound Penalty. Given an original query $(k_o, \vec{w}_o)$ and a missing object $\vec{m}$, we have $r_o = rank(\vec{m}, \vec{w}_o)$. If we simply increase k to r_o, we obtain the penalty $Penalty(k', \vec{w'}) = \lambda$, which can serve as an initial upper bound penalty, denoted by UB_p. Therefore, if the partial penalty caused by Δw of a sampled weighing vector $\vec{w}$ already exceeds UB_p, it can be discarded immediately.

(i) Pruning by Upper Bound Rank. With the current upper bound penalty, the upper bound rank of $\vec{m}$ under the next weighting vector, i.e., UB_r can be determined by:

Lemma 2. *Given a refined query $(k', \vec{w'})$, let the upper bound rank UB_r be defined as follows:*

$$UB_r = k_o + \left\lfloor \left\{ UB_p - (1-\lambda) \frac{\Delta w}{\sqrt{1 + \sum w_o[i]^2}} \right\} \left(\frac{r_o - k_o}{\lambda} \right) \right\rfloor. \quad (9)$$

Then, if $rank(\vec{m}, \vec{w'})$ exceeds UB_r then $(k', \vec{w'})$ cannot be a result, and the progressive top-k process can be ended early.

Proof. A refined query $(k', \vec{w'})$ is worse than the current best refined query if its penalty exceeds the current UB_p. It is obvious that the penalty is controlled by Δk and Δw, and from Equation (4) we can easily compute UB_r by Δw of $(k', \vec{w'})$. □

By adopting the pruning condition in Lemma 2, if $\vec{m}$ does not appear in the top-UB_r results of the progressive query, it can stop early, and $(k', \vec{w'})$ can be safely removed.

(ii) Pruning by Caching Previous Top-k Results. The intuition of this pruning condition is that two similar weighting vectors may have similar top-k results, which can be utilized to skip unnecessary attempts of progressive query.

Lemma 3. *Given a refined query $(k_1, \vec{w}_1)$ that has already been processed with result R_1, and a new refined query $(k_2, \vec{w}_2)$. Comparing $u_{\vec{w}_2}(\vec{m})$ and $u_{\vec{w}_1}(\vec{o})$ for each $\vec{o} \in R_1$, if more than $UB_r - 1$ competitors in R_1 score higher than $u_{\vec{w}_2}(\vec{m})$ among R_1 then $(k_2, \vec{w}_2)$ can be removed.*

Proof. This can be proved by using Lemma 2. □

The progressive top-k algorithm sequentially examines each weighting vector while applying the two pruning rules.

6.2 Branch-and-Bound Algorithm

To address the limitations of the progressive top-k algorithm, we propose a branch-and-bound algorithm that efficiently returns the weighting vector with minimum penalty. Two aggregate R-tree like data structures, *weightRtree* and *compRtree*, are utilized to organize hierarchically the weighting vectors and competitors, respectively. On top of them, several pruning strategies are adopted to reduce the time cost of penalty validation.

6.2.1 Index Structures: weightRtree and compRtree

The *weightRtree* and *compRtree* are both aggregate R-tree like index structures, where *weightRtree* indexes all the weighting vectors and *compRtree* indexes all the competitors of $\vec{m}$ in a space of h derived dimensions.

In the *compRtree*, each entry e_c represents a group of competitors and stores (i) the minimum bounding rectangle (MBR) and (ii) the number of competitors in its subtree, denoted by $N(e_c)$. In the *weightRtree*, each entry e_w stores (i) the MBR of its subtree, (ii) the lower bound Δw of all weighting vectors in its subtree, denoted by $LB_{\Delta w}(e_w)$, and (iii) the upper and lower bound utility of the missing object under all weighting vectors in its subtree, denoted by $UB_{u(\vec{m})}(e_w)$ and $LB_{u(\vec{m})}(e_w)$. Fig. 6 shows an example of a *weightRtree* and a *compRtree*.

6.2.2 Pruning Strategies

We prune the competitors and weighting vectors according to the rank of $\vec{m}$ and the overall penalty.

Upper and Lower Bound Utility of $\vec{m}$ Under e_w. Given an entry e_w and a competitor $\vec{o}$, we are able to derive the upper and lower bound utility in a straightforward manner.

Lemma 4. *Given an entry e_w that covers a set of weighting vectors and a competitor $\vec{o}$, the upper and lower bound utility of $\vec{o}$ under e_w*

$$UB_{u(\vec{o})}(e_w) = \sum_{i=1}^{h} \max_{\vec{w} \in e_w}(w[i]) \cdot o[i]$$

$$LB_{u(\vec{o})}(e_w) = \sum_{i=1}^{h} \min_{\vec{w} \in e_w}(w[i]) \cdot o[i].$$

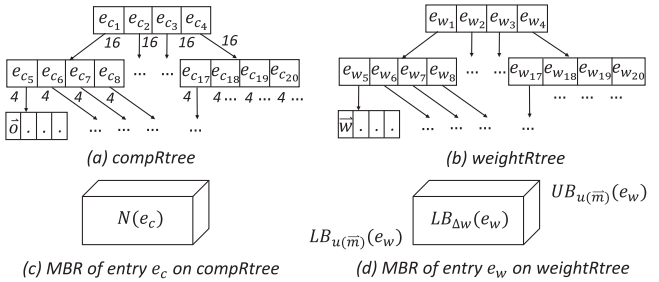


Fig. 6. The *weightRtree* and *compRtree*.

Proof. Assume two artificial weighting vectors $\vec{w}_l$ and $\vec{w}_r$, where $w_l[i] = \min_{\vec{w} \in e_w}(w[i])$ and $w_r[i] = \max_{\vec{w} \in e_w}(w[i])$. Then we have $u_{\vec{w}_l}(\vec{o}) = LB_{u(\vec{o})}(e_w)$ and $u_{\vec{w}_r}(\vec{o}) = UB_{u(\vec{o})}(e_w)$. Therefore it is easy to see that $LB_{u(\vec{o})}(e_w) \leq u_{\vec{w}}(\vec{o}) \leq UB_{u(\vec{o})}(e_w)$ holds for all $\vec{w} \in e_w$. □

This pruning is applied when building the *weightRtree* during precomputation. Basically, we use it to compare the utilities of $\vec{m}$ and $\vec{o}$ under a given e_w. After obtaining the upper and lower bound utility of $\vec{o}$ by Lemma 4, we are able to determine the relative rank of $\vec{o}$ and $\vec{m}$:

(i) if $UB_{u(\vec{o})}(e_w) < LB_{u(\vec{m})}(e_w)$, $\vec{m}$ ranks higher than $\vec{o}$;
(ii) if $LB_{u(\vec{o})}(e_w) > UB_{u(\vec{m})}(e_w)$, $\vec{m}$ ranks lower than $\vec{o}$.

Upper and Lower Bound Utility of e_c Under e_w. Given an entry e_w and e_c, we derive the upper and lower bound utility of e_c under e_w as follows.

Lemma 5. *Given an entry e_c and e_w, the upper and lower bound utility of e_c under e_w can be obtained as follows:*

$$UB_{u(e_c)}(e_w) = \sum_{i=1}^{h} \max_{\vec{w} \in e_w}(w[i]) \cdot \max_{\vec{o} \in e_c}(o[i])$$
$$LB_{u(e_c)}(e_w) = \sum_{i=1}^{h} \min_{\vec{w} \in e_w}(w[i]) \cdot \min_{\vec{o} \in e_c}(o[i]).$$

Proof. Assume two artificial competitors $\vec{o}_l$ and $\vec{o}_r$, where $o_l[i] = \min_{\vec{o} \in e_c}(o[i])$ and $o_r[i] = \max_{\vec{o} \in e_c}(o[i])$. From Definition 5 and Observation 1, we know that any $\vec{o} \in e_c$ dominates $\vec{o}_l$ and is dominated by $\vec{o}_r$. From Lemma 4, we obtain the upper and lower bound utility of $\vec{o}$ under e_w. Therefore, the upper bound utility of e_c under e_w is bounded by that of o_r under e_w. Likewise, the lower bound utility of e_c under e_w is bounded by that of o_l under e_w. □

Rank Range of $\vec{m}$ Under e_w. With Lemmas 4 and 5, we are able to derive the range of $rank(\vec{m}, e_w)$, which can be utilized for pruning.

Lemma 6. *For an entry e_w and e_c, if $LB_{u(\vec{m})}(e_w)$ is greater than $UB_{u(e_c)}(e_w)$, we have $rank(\vec{m}, e_w) \leq c - N(e_c) + 1$. Likewise, if $UB_{u(\vec{m})}(e_w)$ is smaller than $LB_{u(e_c)}(e_w)$, we have $rank(\vec{m}, e_w) \geq N(e_c) + 1$.*

Proof. If $LB_{u(\vec{m})}(e_w) \geq UB_{u(e_c)}(e_w)$ holds, then $\vec{m}$ ranks higher than $N(e_c)$ competitors; If $UB_{u(\vec{m})}(e_w) < LB_{u(e_c)}(e_w)$, then $\vec{m}$ ranks lower than $N(e_c)$ competitors. Thus it is proved. □

Penalty Based Pruning. Given an entry e_w, assume the range of $rank(\vec{m}, e_w)$ has already been obtained. As we can obtain $LB_{\Delta w}(e_w)$ directly from the *weightRtree*, the lower bound penalty $LB_p(e_w)$ can be computed easily.

Lemma 7. *If $LB_p(e_w)$ exceeds current upper bound penalty UB_p, then e_w can be discarded safely.*

Observation 3. For an entry e_w, based on Lemmas 2 and 7, we can derive a UB_r from Equation (9). Thus, we do not need to determine the rank range of $rank(\vec{m}, e_w)$ exactly. Instead, once more than UB_r competitors are seen to score higher than $\vec{m}$, e_w can be removed.

6.2.3 Search Algorithm

The algorithm involves two levels of branch-and-bound search. Before introducing the BAB algorithm, we present a procedure ComputeLBR() that determines the lower bound of $rank(\vec{m}, e_w)$, i.e., $LBR(\vec{m}, e_w)$. With ComputeLBR(), we are able to derive the lower bound penalty of the missing object under a set of weighting vectors and decide if we can prune them by comparing with the current UB_p. In ComputeLBR(), we initialize an empty heap H to keep entries e_c from the *compRtree* based on their $LB_{u(e_c)}(e_w)$ and conduct a best-first search to examine the entries. Given an entry e_w, we insert the root entry of the *compRtree* into H. In each round, we pop the top entry e_c of H and apply operations based on its type. The details are presented in Algorithm 3.

(i) *Pruning at non-leaf nodes.* For each subnode e'_c of e_c, we first compute $LB_{u(e'_c)}(e_w)$ and compare with $UB_{u(\vec{m})}(e_w)$. From Lemma 6, if $LB_{u(e'_c)}(e_w)$ is no smaller than $UB_{u(\vec{m})}(e_w)$, we update $LBR(\vec{m}, e_w)$ by adding $N(e'_c)$. Otherwise, we insert e'_c into H.
(ii) *Pruning at leaf nodes.* For each $\vec{o}$ in e_c, we compute $LB_{u(\vec{o})}(e_w)$ and compare with $UB_{u(\vec{m})}(e_w)$. From Lemma 4, if $LB_{u(\vec{o})}(e_w)$ is no smaller than $UB_{u(\vec{m})}(e_w)$, we update $LBR(\vec{m}, e_w)$ by adding 1.

Algorithm 3. Procedure ComputeLBR()

Input: e_w, *compRtree* on C and UB_r
Output: $LBR(\vec{m}, e_w)$

```
1: Initialize H and insert the root of compRtree into H;
2: while H is not empty do
3:   Pop the top entry e_c of H;
4:   if e_c is a leaf node then
5:     foreach o ∈ e_c do
6:       Compute LB_{u(o)}(e_w);
7:       if LB_{u(o)}(e_w) ≥ UB_{u(m)}(e_w) then
8:         Update LBR(m, e_w) by adding 1;
9:         if LBR(m, e_w) > UB_r then
10:          return false;
11:  else
12:    foreach e'_c ∈ e_c do
13:      Compute LB_{u(e'_c)}(e_w);
14:      if LB_{u(e'_c)}(e_w) ≥ UB_{u(e'_c)}(e_w) then
15:        Update LBR(m, e_w) by adding N(e'_c);
16:        if LBR(m, e_w) > UB_r then
17:          return false;
18: return LBR(m, e_w);
```

Terminate Condition. From Observation 3, if the current $LBR(\vec{m}, e_w)$ already exceeds UB_r, ComputeLBR() stops, and all weighting vectors in e_w are eliminated from consideration as a result. Otherwise, it stops when H is empty and returns the current lower bound rank $LBR(\vec{m}, e_w)$.

It is worth noting that ComputeLBR() returns the exact $rank(\vec{m}, \vec{w})$ if the input contains only one $\vec{w}$. Algorithm BAB uses ComputeLBR(). The algorithm keeps an empty heap H to store entries e_w from the *weightRtree*, and the entries in H

TABLE 2
Statistics of Dataset

	Beijing	New York	Synthetic
# of PoIs	329,481	206,416	1,000,000
# of keywords	88,190	47,394	80,000
Avg occurrence of a keyword	22	35	75
Avg # of keywords per PoI	6	8	6

are sorted based on their $LB_p(e_w)$ value. We initialize H by inserting the root node of the *weightRtree* into H and set the initial $UB_p = \lambda$. Each e_w or $\vec{w}$ that renders the partial penalty of Δw higher than the current UB_p is discarded immediately. The traversal of the *weightRtree* terminates when H becomes empty.

(i) *Pruning at non-leaf nodes.* For each subnode e'_w of e_w, we first compute UB_r with $LB_{\Delta w}(e'_w)$ and the current UB_p based on Observation 3 and apply ComputeLBR() on it. If $LBR(\vec{m}, e'_w)$ is successfully returned, we compute $LB_p(e'_w)$ and insert e'_w into H.

(ii) *Pruning at leaf nodes.* For each $\vec{w}$ in e_w, we first compute UB_r with Δw and the current UB_p using Equation (9). If ComputeLBR() successfully returns $rank(\vec{m}, \vec{w})$, we update UB_p by $Penalty(rank(\vec{m}, \vec{w}), \vec{w})$.

Algorithm 4. Branch-and-Bound Algorithm (BAB)

Input: *weightRtree* on W and *compRtree* on C
Output: The best refined query $(k_b, \vec{w}_b)$
1: Initialize H, UB_p and insert the root of *weightRtree* into H;
2: **while** H *is not empty* **do**
3: Pop the top entry e_w of H;
4: **if** e_w *is a leaf node* **then**
5: **foreach** $\vec{w} \in e_w$ **do**
6: Compute UB_r with Δw and UB_p;
7: **if** ComputeLBR() is true **then**
8: Update UB_p and $(k_b, \vec{w}_b)$;
9: **else**
10: **foreach** $e'_w \in e_w$ **do**
11: Compute UB_r with $LB_{\Delta w}(e'_w)$ and UB_p;
12: **if** ComputeLBR() *is true* **then**
13: Add e'_w into H;
14: **return** UB_p *and* $(k_b, \vec{w}_b)$;

In the worst case, BAB has to enumerate all the competitors and weights stored in *compRtree* and *weightRtree*. In practice, BAB runs fast due to the pruning power.

7 Experiments

We report on extensive experiments with real geo-textual datasets that offer insight into the performance of the proposed index structures and algorithms.

7.1 Experimental Settings

All algorithms were implemented in GNU C++ on Linux and run on an Intel(R) CPU i7-4770@3.4 GHz and 32G RAM.

Datasets. We use two real PoI datasets, Beijing PoI and New York PoI, that consist of keywords of PoIs from the OpenStreetMap[5] and Foursquare,[6] and a synthetic dataset.

5. https://www.openstreetmap.org
6. https://foursquare.com/

TABLE 3
Parameter Settings

Parameters	Values
Dataset cardinality	20K, 40K, 80K, 160K, **320K**
λ	0.1, 0.3, **0.5**, 0.7, 0.9
# of users h	**3**, 4, 5, 6, 7
the diameter of Q (km)	5, **10**, 20, 30, 50
k	5, **10**, 20, 50, 100
Actual rank of $\vec{m}$	11, **101**, 501, 1001
$T\%$	0.3%, 0.25%, **0.2%**, 0.15%, 0.1%
Pr	0.5, 0.6, **0.7**, 0.8, 0.9
# of missing objects $\lvert M\rvert$	**1**, 3, 5, 7
# of query keywords	**1**, 2, 3, 4, 5

TABLE 4
Performance of Algorithms on QT (Query Time (ms)) and P (Penalty)

Algo		GTD	RSA+PTK		ISA-US +BAB		ISA-WM +BAB		ISA-RI +BAB	
QT	BJ	198	277	1701	347	680	365	377	394	264
	NY	133	238	1299	278	645	284	389	319	176
	SYN	682	372	3421	424	2241	519	1256	592	772
P	BJ	N/A	0.31		0.20		0.18		**0.15**	
	NY	N/A	0.42		0.28		0.22		**0.17**	
	SYN	N/A	0.51		0.32		0.25		**0.23**	

Each PoI has a name, a location (in the form of longitude and latitude), and category tags (with several subcategories). We combine the name and categories of a PoI to form its textual information of each PoI. As shown in Table 2, for Beijing, we have 329,481 PoIs and 88,190 keywords, and the average occurrence of a keyword is 21. For New York, we have 206,416 PoIs and 87,394 keywords, and the average occurrence of a keyword is 18. As these two real datasets are similar in terms of scale and keyword distribution, we only present the performance on the Bejing and synthetic datasets due to the space limitation.

Algorithms. We evaluate the performance of the proposed algorithms: geo-textual dominance search algorithm (GTD), the incremental sampling algorithm (ISA), and the branch-and-bound algorithm (BAB). In ISA, we use three different heuristic strategies, i.e., ISA-US, ISA-WM, ISA-RI, and compare with the baseline RSA. We set η in ISA-RI to 100. For BAB, we evaluate the performances by comparing with the baseline progressive top-k algorithm (PTK).

Parameter Settings. We randomly generate 100 queries for each experiment and report their average performance. To evaluate the algorithms in different settings, we vary the values of parameters, as shown in Table 3. As default settings, we choose 320 K for the dataset cardinality, 0.5 for λ, 3 for the number of users, 10 km for the diameter of query locations, 10 for the top-k, 101 for the actual rank of $\vec{m}$, 2 percent for $T\%$, 0.7 for Pr, 1 for the number of missing objects and 1 for the number of query keywords. For the original query $(k_o, \vec{w}_o)$, we set $\vec{w}_o = (1/h, \ldots, 1/h)$ and $k_o = k$.

7.2 Performance Evaluation

Table 4 compares the proposed algorithms with respect to query time and penalty by using the default settings. For the penalty, we can see that ISA generate weighting

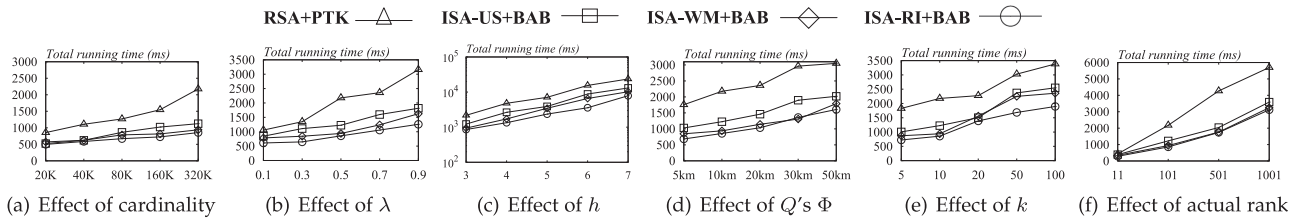


Fig. 7. Effect of varying parameters, Beijing dataset.

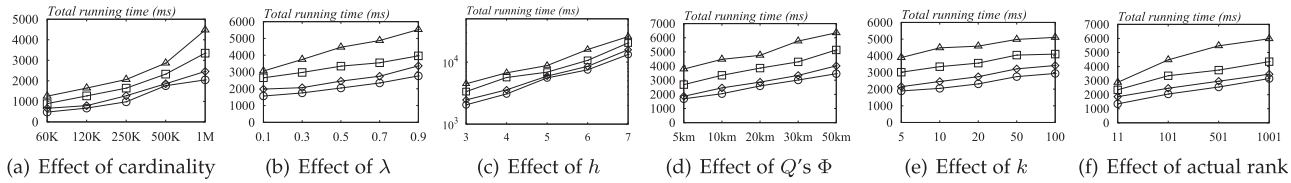


Fig. 8. Effect of varying parameters, synthetic dataset.

vectors with higher quality than RSA, especially when using the RI method. Thus, the penalty is smaller and closer to optimal result. For the query time, we can see that BAB algorithm reduces the time of validating refined queries, and even if ISA takes more time than RSA for sampling, ISA-RI + BAB still takes the least overall query time. In the remaining experiments, we only show the performance on the Beijing and synthetic datasets since the performance on the Beijing and New York datasets are similar.

Effect of Dataset Cardinality. We study the effect of the dataset cardinality on the performance of algorithms RSA + PTK, ISA-US + BAB, ISA-WM + BAB, and ISA-$RI + BAB$. We sample 4 datasets from the Beijing dataset with 20 to 160 K PoIs, and 4 datasets from the synthetic dataset with 60 to 500z K. As can be seen in Figs. 7a and 8a, our algorithms scale linearly with the dataset cardinality. In general, the ISA + BAB algorithm outperforms the baseline RSA + PTK increasingly when the we enlarge the cardinality. A possible explanation is that when more competitors are involved, the computeLBR() speeds up the query by efficiently pruning weighting vectors.

Effect of λ. We study the effect of λ on the performance. We choose 5 values of λ from 0.1 to 0.9. As we can see in Figs. 7b and 8b, the query takes more time when λ increases. As we mentioned in Section 6.1, the initial upper bound penalty is set to λ. Therefore, when λ is smaller, the pruning power is stronger. However, the performance of the algorithms do not differ much.

Effect of the Number of Users h. We study the effect of the number of users on the performance by varying h from 3 to 7. As we know, h is the dimensionality of the objects and the weighting vectors. In Figs. 7c and 8c we can see that the performance of all the algorithms degrades with the growth of the dimensionality. This is because all the algorithms need to traverse the IR^2-tree by using GTD, and BAB needs to traverse the *compRtree* and the *weightRtree*. However, our proposed methods still outperform the baseline.

Effect of Q's Diameter. We examine the effect of the distribution of query locations. The diameter of the query, which is the maximum distance among all pairs of query locations. As can be seen in Figs. 7d and 8d, the query time increases when we enlarge the diameter from 5 to 50 km. This occurs because more objects are involved when the area increases, which affects the number of competitors and the weighting vectors in the query.

Effect of k. We study the effect of k on the performance by varying k from 5 to 100. For instance, given a top-5 GSK query, the corresponding WGSK looks for the missing object ranked 51th. Thus, a higher k increases the time needed for traversing R-tree. Moreover, a worse rank of the missing object consumes more time in each step of the solution. As shown in Figs. 7e and 8e, our proposed algorithms scale well with the increase of k compared with the baseline.

Effect of $\vec{m}$'s Actual Rank. We study the effect of $\vec{m}$'s actual rank on the performance. We use the default setting of $k = 10$ and vary $\vec{m}$'s actual rank from 11 to 1,001. Not surprisingly, in Figs. 7f and 8f, the proposed algorithms outperform the baseline and scale well. As mentioned in the last experiment, the penalty validation takes more time when the missing object has a worse rank since we need to determine the rank under a set of candidate weighting vectors.

Effect of $T\%$ and Pr. We study the effect by varying $T\%$ from 0.3 to 0.1 percent, and by varying Pr from 0.5 to 0.9. These two parameters are used to control the quality of the weighting vectors we sampled. In addition to the efficiency evaluation, we also study the effectiveness of our algorithms with different values of $T\%$ and Pr. As the same in [18], we use the penalty to reflect the effectiveness, where a lower and convergent penalty indicates a higher degree of approximation to the optimal result. Figs. 9a and 9c show the query time and penalty with different quality guarantee. When enlarging $T\%$, more weighting vectors are sampled, and the query time increases since steps 2 and 3 involve more computation. Moreover, we notice that the penalty decreases and tends to converge as $T\%$ increases. The ISA-RI + BAB outperforms ISA-US + BAB and ISA − WM + BAB since it achieves both lower penalty and less running time. As we can see in Figs. 9b and 9d the query time increases and penalty decreases. As both $T\%$ and Pr are used for the quality control, the trends are similar and converge gradually. The results for the synthetic dataset are similar to those for the Beijing dataset, as shown in Fig. 10.

Effect of Multiple Query Keywords. We study the effect of multiple query keywords on the performance by varying the number of keywords from 1 to 5. We consider the disjunctive case where an object that contains any query keyword is a candidate for the GSK query result. It is easy to see that this only increases the number of competitors in Step (i). The rest of the solution is the same. Thus, more keywords means

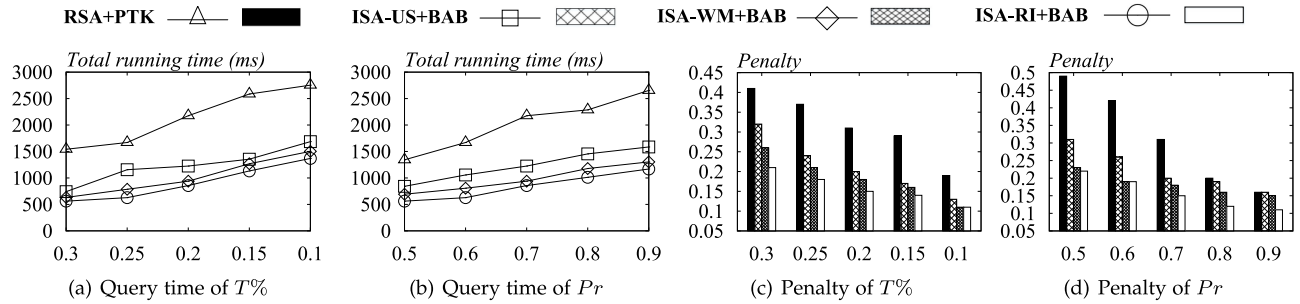


Fig. 9. Effect of $T\%$ and Pr on query time (ms) and penalty, Beijing dataset.

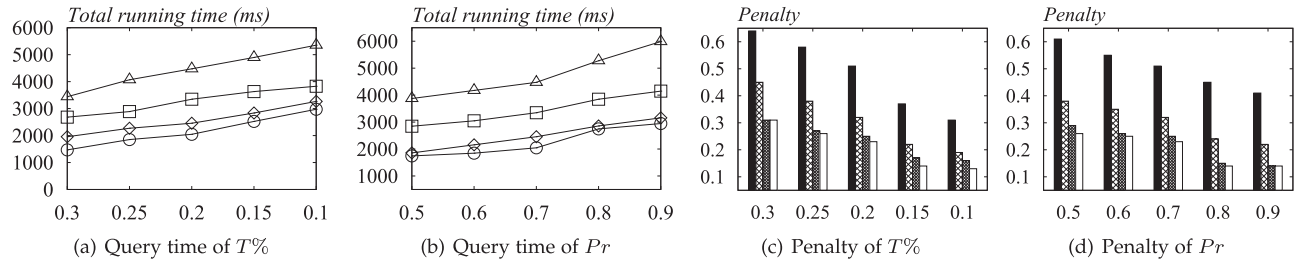


Fig. 10. Effect of $T\%$ and Pr on query time (ms) and penalty, synthetic dataset.

more candidate objects. In Figs. 11a and 12a, our methods again outperform the baseline.

Effect of Multiple Missing Objects. We study the effect of multiple missing objects on the performance by considering 1, 3, 5, and 7 objects. To cope with multiple missing objects M, we assume there is no dominance relationship between any two missing objects. In Step (i), we extract the competitors for all missing objects. In Step (ii), we sample the weighting vectors on the partition hyperplane between the competitor and the corresponding missing object. In Step (iii), we only need to consider the missing object with minimum utility or lower bound utility among all $\vec{m} \in M$ in computeLBR(). As we can see in Figs. 11b and 12b, the time increases when more missing objects are considered. This is because more competitors can be obtained and because we have to consider the missing object with the worst rank.

8 Related Work

8.1 Spatial Keyword Search

Searching geo-textual objects based on a query location and keywords has attracted substantial attention. In euclidean space, the *IR*^{2}*-tree* [27] integrates signature files and the *R-tree* to answer Boolean keyword queries. The *IR-tree* [28] is an *R-tree* augmented with inverted files that supports the ranking of objects based on a scoring function that involves spatial distance and text relevancy. A recent study [29] provides a survey of twelve state-of-the-art geo-textual indices and presents a performance comparison of the indices. Cao et al. [3] propose a collective spatial keyword query, that returns a group of objects whose textual descriptions cover given query keywords and ranks the highest according to spatial criteria, such as having the smallest sum of distances to a query location. ROAD [30] organizes a road network as a hierarchy of subgraphs and connects these using shortcuts. For each subgraph, an object abstract is generated for keyword checking. By using network expansion, the subgraphs without intended objects are pruned. The G-tree [31] adopts a graph partitioning approach to form a hierarchy of subgraphs. Within each subgraph, a distance matrix is kept, and for any two subgraphs, the distances between their borders are stored as well. Based on these distances, the distance between a query vertex and target vertices or tree nodes can be computed efficiently.

8.2 Why-Not Queries

To answer why-not questions, Huang et al. [17] first explored the provenance of non-answers. Extensive efforts have subsequently been put into answering why-not questions. The existing approaches can be classified into three categories: (i) manipulation identification (e.g., why-not questions on SPJ queries [14] and SPJUA queries [15]), (ii)

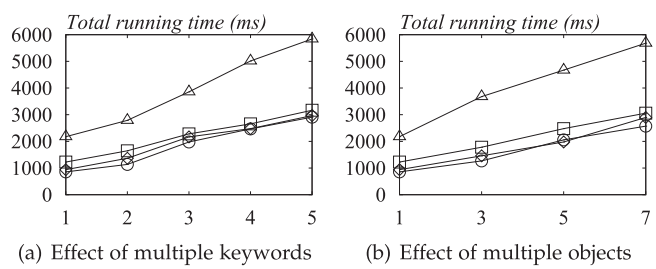


Fig. 11. Effect of multiple keywords and objects, Beijing dataset.

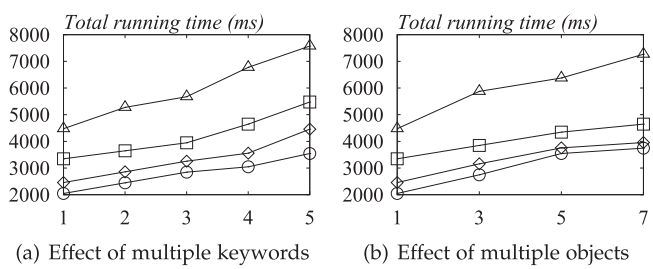


Fig. 12. Effect of multiple keywords and objects, synthetic dataset.

database modification (e.g., why-not questions on SPJ queries [17]), and (iii) query refinement (e.g., why-not questions on SPJA queries [16], top-k queries [18], reverse skyline queries [32], spatial keyword top-k queries [19], and metric probabilistic range queries [33]). In addition, Herschel [34] tries to identify hybrid why-not explanations for SQL queries. Next Ten Cate et al. [35] present a framework for why-not explanations by leveraging concepts from an ontology. Bidoit et al. [36] provide a new formalization of why-not explanations as polynomials. Liu et al. [37] conduct an excellent work on answering why-not and why questions on reverse top-k queries. In contrast, we offer the first study of why-not query processing for the relatively complex top-k group spatial keyword query. He et al. [18] study a related problem on minimizing the overall change of $\vec{w}$ and k while achieving the inclusion. However, their solutions can only work for static datasets, and do not apply to spatial keyword queries where query locations are dynamic and precomputation based on spatial distance is infeasible. Chen et al. [19] answer the *why-not* questions on top-k spatial keyword queries, which is the most related work, but their changing on $\vec{w}$ is limited to spatial and textual dimensions only, which can not be applied to high dimensions.

9 Conclusion and Future Work

Given the result of a top-k group spatial keyword query, users may wonder why the result fails to include an expected answer. In this setting, the why-not group spatial keyword query is able to return the result of minimally modified query that does include the expected answer. We present a framework that provides a three-phase solution to computing such why-not queries. We adopt query refinement to modify the users preference $\vec{w}$ and parameter k in the original top-k query so that the expected answer is included in the result of the modified query. The first phase extracts competitors from the data set. The second phase generates candidate weightings to reform promising queries. Finally, we return the query that is most similar to the original query. Several directions for future research are promising. First, the query model can be extended to road networks, enabling users to search for geo-textual object while considering the network distance. Second, the top-k group spatial keyword query only considers the spatial proximity between the object and users. Extensions to support additional aspects, such as popularity, rating, price, and the number of reviews are of interest.

Acknowledgments

This research is partially supported by NSFC (Grants No. 61532018, 61836007, 61832017, and 61572215) and a grant from the Obel Family Foundation and the DiCyPS project.

References

[1] B. Zheng, N. J. Yuan, K. Zheng, X. Xie, S. Sadiq, and X. Zhou, "Approximate keyword search in semantic trajectory database," in *Proc. IEEE Int. Conf. Data Eng.*, 2015, pp. 975–986.

[2] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang, "Towards efficient search for activity trajectories," in *Proc. IEEE Int. Conf. Data Eng.*, 2013, pp. 230–241.

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

[4] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu, "Collective spatial keyword queries: A distance owner-driven approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 689–700.

[5] T. Guo, X. Cao, and G. Cong, "Efficient algorithms for answering the m-closest keywords query," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 405–418.

[6] D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Proc. IEEE Int. Conf. Data Eng.*, 2009, pp. 688–699.

[7] J. Han, A. Sun, G. Cong, W. X. Zhao, Z. Ji, and M. C. Phan, "Linking fine-grained locations in user comments," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 1, pp. 59–72, Jan. 2018.

[8] K. Deng, S. Sadiq, X. Zhou, H. Xu, G. P. C. Fung, and Y. Lu, "On group nearest group query processing," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 2, pp. 295–308, Feb. 2012.

[9] J. Zhang, W.-S. Ku, M.-T. Sun, X. Qin, and H. Lu, "Multi-criteria optimal location query with overlapping voronoi diagrams," in *Proc. Int. Conf. Extending Database Technol.*, 2014, pp. 391–402.

[10] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *Proc. IEEE Int. Conf. Data Eng.*, 2004, pp. 301–312.

[11] M. L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 820–833, Jun. 2005.

[12] K. Deng, X. Zhou, and H. Tao, "Multi-source skyline query processing in road networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2007, pp. 796–805.

[13] J. Shi, D. Wu, and N. Mamoulis, "Textually relevant spatial skylines," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 224–237, Jan. 2016.

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

[15] N. Bidoit, M. Herschel, and K. Tzompanaki, "Query-based why-not provenance with NedExplain," in *Proc. Int. Conf. Extending Database Technol.*, 2014, pp. 145–156.

[16] M. Herschel and M. A. Hernández, "Explaining missing answers to SPJUA queries," *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 185–196, 2010.

[17] 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, 2008.

[18] Z. He and E. Lo, "Answering why-not questions on top-k queries," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 6, pp. 1300–1315, Jun. 2014.

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

[20] K. Mouratidis, Y. Lin, and M. L. Yiu, "Preference queries in large multi-cost transportation networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2010, pp. 533–544.

[21] A. Yu, P. K. Agarwal, and J. Yang, "Top-k preferences in high dimensions," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 2, pp. 311–325, Feb. 2016.

[22] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, "Answering top-k queries using views," in *Proc. Int. Conf. Very Large Data Bases*, 2006, pp. 451–462.

[23] L. Zou and L. Chen, "Dominant graph: An efficient indexing structure to answer top-k queries," in *Proc. IEEE Int. Conf. Data Eng.*, 2008, pp. 536–545.

[24] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Comput. Surveys*, vol. 40, no. 4, pp. 11:1–11:58, 2008.

[25] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. IEEE Int. Conf. Data Eng.*, 2001, pp. 421–430.

[26] L. D. Berkovitz, *Convexity and Optimization in Rn*. Hoboken, NJ, USA: Wiley, 2003.

[27] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. IEEE Int. Conf. Data Eng.*, 2008, pp. 656–665.

[28] 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, 2009.

[29] L. 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.

[30] K. C. Lee, W.-C. Lee, and B. Zheng, "Fast object search on road networks," in *Proc. Int. Conf. Extending Database Technol.*, 2009, pp. 1018–1029.

[31] R. Zhong, G. Li, K.-L. Tan, and L. Zhou, "G-Tree: An efficient index for kNN search on road networks," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 39–48.
[32] Q. Liu, Y. Gao, G. Chen, B. Zheng, and L. Zhou, "Answering why-not and why questions on reverse top-k queries," *VLDB J.*, vol. 25, no. 6, pp. 867–892, 2016.
[33] L. Chen, Y. Gao, K. Wang, C. S. Jensen, and G. Chen, "Answering why-not questions on metric probabilistic range queries," in *Proc. IEEE Int. Conf. Data Eng.*, 2016, pp. 767–778.
[34] M. Herschel, "Wondering why data are missing from query results?: Ask conseil why-not," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 2213–2218.
[35] B. T. Cate, C. Civili, E. Sherkhonov, and W.-C. Tan, "High-level why-not explanations using ontologies," in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2015, pp. 31–43.
[36] N. Bidoit, M. Herschel, and K. Tzompanaki, "EFQ: Why-not answer polynomials in action," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1980–1983, 2015.
[37] Q. Liu, Y. Gao, G. Chen, B. Zheng, and L. Zhou, "Answering why-not and why questions on reverse top-k queries," *VLDB J.*, vol. 25, no. 6, pp. 867–892, 2016.

Bolong Zheng received the bachelor's and master's degrees in computer science from the Huazhong University of Science and Technology, in 2011 and 2013, respectively, and the PhD degree from the University of Queensland, in 2017. He is an associate professor with the Huazhong University of Science and Technology (HUST). His research interests include spatio-temporal data management and graph data management.

Kai Zheng received the PhD degree in computer science from the University of Queensland, in 2012. He is a full professor with the University of Electronic Science and Technology of China. He has been working in the area of social-media analysis, spatial-temporal databases, and uncertain databases. He has published more than 70 papers in the most prestigious journals and conferences such as SIGMOD, ICDE, the *VLDB Journal*, ACM transactions and IEEE transactions.

Christian S. Jensen is an Obel professor of computer science with Aalborg University, Denmark. He was a professor with Aarhus University for a 3-year period from 2010 to 2013, and he was previously with Aalborg University for two decades. He recently spent a 1-year sabbatical with Google Inc., Mountain View, CA. His research concerns data management and data-intensive systems, and its focus is on temporal and spatio-temporal data management. He is a member of the Academia Europaea, the Royal Danish Academy of Sciences and Letters, and the Danish Academy of Technical Sciences. He has received several national and international awards for his research. He is editor-in-chief of the *ACM Transactions on Database Systems* and was an editor-in-chief of the *VLDB Journal* from 2008 to 2014. He is a fellow of the IEEE and ACM.

Nguyen Quoc Viet Hung received the master's and PhD degrees from EPFL, Switzerland, under the supervision of Prof. Karl Aberer. He is a lecturer with Griffith University. He spent two years as a postdoc with EPFL and two years as a postdoc with UQ. His research focuses on data integration, data quality, information retrieval, trust management, recommender systems, machine learning, and big data visualization, with special emphasis on web data, social data, and sensor data. He published papers in top-tier venues such as SIGMOD, SIGIR, ICDE, IJCAI, ICDM, the *VLDB Journal*, and the *IEEE Transactions on Knowledge and Data Engineering*.

Han Su received the BS degree in software engineering from Nanjing University, in 2011, and the PhD degree in computer science from the University of Queensland, in 2015. She is currently an associate professor with the Big Data Research Center, University of Electronic Science and Technology of China. Her research interests include trajectory querying and mining.

Guohui Li is a full professor with the School of Software Engineering, Huazhong University of Science and Technology (HUST). His research interests include big data processing, data mining, social media data processing, and real-time computing. He is widely published in the leading scholarly journals and conferences in computer science, such as RTSS, AAAI, the *IEEE Transactions on Computers*, and the *IEEE Transactions on Mobile Computing*.

Xiaofang Zhou received the bachelor's and master's degrees in computer science from Nanjing University, in 1984 and 1987, respectively, and the PhD degree in computer science from the University of Queensland, in 1994. He is a professor of computer science with the University of Queensland. He is the head of the Data and Knowledge Engineering Research Division, School of Information Technology and Electrical Engineering. He is also a specially appointed adjunct professor with Soochow University, China. His research is focused on finding effective and efficient solutions to managing integrating, and analyzing very large amounts of complex data for business and scientific applications. His research interests include spatial and multimedia databases, high performance query processing, web information systems, data mining, and data quality management. He is a fellow of the IEEE.