2016

Answering Why-Not Spatial Keyword Top-k Queries via Keyword Adaption

Jianliang Xu
Computer Science Department, Hong Kong Baptist University

Lei Chen
Computer Science Department, Hong Kong Baptist University

Xin Lin
Shanghai Key Laboratory of Multidimensional Information Processing, East China Normal University, China

Christian S. Jensen
Department of Computer Science, Aalborg University, Denmark

Haibo Hu
Department of Electronic and Information Engineering, Hong Kong Polytechnic University

Answering Why-Not Spatial Keyword Top-k Queries via Keyword Adaption

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

***Abstract*—Web objects, often associated with descriptive text documents, are increasingly being geo-tagged. A spatial keyword top-k query retrieves the best k such objects according to a scoring function that considers both spatial distance and textual similarity. However, it is in some cases difficult for users to identify the exact keywords that describe their query intent. After a user issues an initial query and gets back the result, the user may find that some expected objects are missing and may wonder why. Answering the resulting why-not questions can aid users in retrieving better results. However, no existing techniques are able to answer why-not questions by adapting the query keywords. We propose techniques capable of adapting an initial set of query keywords so that expected, but missing, objects enter the result along with other relevant objects. We develop a basic algorithm with a set of optimizations that sequentially examines a sequence of candidate keyword sets. In addition, we present an index-based bound-and-prune algorithm that is able to determine the best sample out of a set of candidates in just one pass of index traversal, thus speeding up the query processing. We also extend the proposed algorithms to handle multiple missing objects. Extensive experimental results offer insight into the efficiency of the proposed techniques in terms of running time and I/O cost.**

I. Introduction

With the proliferation of geo-enabled mobile devices, notably smartphones, location-based services that target web objects with geo-location and textual descriptions, *e.g.*, businesses and public facilities, are gaining in prominence. In particular, so-called *spatial keyword queries* [3] enable a range of services that retrieve such objects. More specifically, a spatial keyword top-k query takes a user location and a set of keywords as arguments and retrieves the k objects that are ranked the highest according to a scoring function that considers both spatial distance and textual similarity [10].

However, it can be difficult for users to identify the keywords that best capture the intent of their queries. Thus, having issued a query with a set of query keywords and having received the result, a user may find that the result is not as expected. Specifically, objects that the user expected to be in the result are missing. This suggests to the user that other useful objects, that are as yet unknown to the user, may be missing from the result as well, and the user has reason to question the overall utility of the query and its result.

In this setting, the utility of spatial keyword querying can be improved by offering functionality that explains to the user why one or more expected objects are missing and how to minimally modify the initial query so that the missing objects, and then potentially also other useful objects, become part of the result. This paper considers such functionality, as illustrated in the following example.

Example 1: In preparation for attending an overseas conference, a user issues a query to find the top-3 hotels that are close to the conference venue and are described as "clean" and "comfortable." The user is surprised that the result contains only local hotels that are unknown to the user and that a well-known nearby international hotel is not in the result. The user wonders why this exclusion happens. Are the returned hotels really the best, or are there better options? Are the query keywords not properly set? How can the keywords be adapted so that the expected hotels appear in the result?

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

Example 2: Consider a user who opens a Sichuan restaurant near the Oriental Pearl Tower in Shanghai, China. The user lists the restaurant in online catalogs such as Google My Business. To attract more customers, the user wants to advertise the keywords in which the restaurant ranks high when the customers search the catalog near the Oriental Pearl Tower. However, the simple keywords "Sichuan cuisine" do not return the restaurant as a top-10 result. The user wants to know why and how the keywords should be modified so that the restaurant can enter the top-10.

These scenarios call for support for *why-not* questions, which were first introduced by Chapman and Jagadish [6]. Three different solution models have been proposed to answer such why-not questions: *manipulation identification* [6], *database modification* [16], [17], and *query refinement* [8], [15], [28]. Using manipulation identification, Chapman and Jagadish [6] study why-not functionality for Select-Project-Join (SPJ) queries, showing how to find a query operator that prevents a missing object from being included in a result. Studies using database modification [16], [17] consider how to update the original database so that SPJ queries and SPJ + Union + Aggregation queries can revive missing objects. In contrast, studies based on query refinement [28] focus

on how to revise an original query so that a missing object enters the result. He and Lo [15] apply this approach to top-k preference queries. In previous work [8], we consider why-not questions on spatial-keyword top-k queries to help users adjust their preferences between spatial distance and textual similarity to obtain query results that contain missing objects. However, this past study does not provide users with more precise keywords that better describe their query intention.

In this paper, we apply the *query refinement* model to solve the problem of *answering why-not questions on spatial keyword top-k queries via keyword adaption*. The paper's contributions are summarized as follows:

- We formulate the novel keyword-adapted why-not spatial keyword top-k query and reduce it to a query refinement problem.
- We propose a basic algorithm as well as a set of optimizations to efficiently search the candidate query keyword sets to find the optimal solution.
- We propose a more advanced algorithm based on an index structure, which determines the best keyword set among a set of candidates using a bound-and-prune strategy.
- We extend the algorithms to support queries with multiple missing objects and provide strategies that find approximate results and balance solution quality and query response time.
- We report on extensive experiments on real-life datasets to evaluate the performance of the proposed algorithms. The results show that the algorithms perform well in a wide range of settings.

The rest of the paper is organized as follows. Section II reviews related work. Section III describes preliminaries and defines the keyword-adapted why-not spatial keyword top-k query. Section IV presents the basic query algorithm along with a set of optimizations. Section V proposes an index-based solution. We extend the algorithm to support queries with multiple missing objects in Section VI. Experimental results are reported in Section VII, followed by a conclusion in Section VIII.

II. Related Work

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

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. A number of efficient query processing techniques have been proposed for this query. Early work presents a hybrid index structure that integrates R*-tree and inverted file for the estimation of both spatial and textual similarities [34]. Martins *et al*. [25] compute text relevancy and location proximity independently and then combine the two aspects. The more recent IR-tree [10], [20], [29] is a hybrid index that estimates the bounds of spatial distance and textual similarity at the same time. Another hybrid index, the IR^2-tree [11] combines an R-tree [14] with superimposed text signatures. However, this index is applicable only when the keywords serve as a Boolean filter. Rocha-Junior and Nørvåg [27] study the spatial keyword query in road networks. A comprehensive experimental evaluation of different spatial keyword indexing and query processing techniques is available [7].

Different variants of the spatial keyword query have also been considered. Chen *et al*. [9] study a query that retrieves web pages which contain query keywords and whose page footprints intersect with a query footprint. A recently proposed mCK query retrieves the best m objects within a minimum diameter that match given keywords. The bR*-tree and the virtual bR*-tree [32], [33], which augment each node with a bitmap and MBRs for keywords, are proposed for computing this query. Cao *et al*. [4] introduce a query that retrieves the top-k spatial web objects ranked according to both prestige-based relevance and location proximity. Another study [5] proposes a query that retrieves a group of nearby spatial web objects whose keywords cover the query's keywords and that have the lowest inter-object distances. Further, spatial keyword similarity search in regions of interest has been studied [12]. Other studies consider a direction-aware spatial keyword query that finds k nearest neighbors in the search direction that cover all query keywords [19]. Li *et al.* [21] investigates a spatial approximate string query that is a range query augmented with a string similarity predicate. Bouros *et al.* [2] aim to identify pairs of objects from a spatio-textual database that are both spatially close and textually similar. Another study [30] integrates the social influence into traditional spatial keyword search to improve the answer quality. However, none of the work mentioned above 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 introduced by Chapman and Jagadish [6]. Existing approaches can be classified into three categories. Chapman and Jagadish use *manipulation identification* to identify operations that filter out missing objects. Other studies [16], [17] adopt a *database modification* to update an original database so that missing objects become part of query results. Tran and Chan [28] retrieve the missing objects through *query refinement*. He and Lo [15] 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 recent, studies [1], [13], [18] consider how to answer why-not questions using query refinement in the contexts of social image search, reverse skyline queries, and reverse top-k queries, respectively. In [8], we study the why-not question on spatial keyword top-k queries. However, it only adjusts the users' preferences between the spatial distance and the textual similarity. It does not suggest keywords that better describe users' query intentions, which is the focus of this paper. As adapting the query keywords needs to re-evaluate the textual similarity between the query and objects, the existing techniques are not applicable here.

III. Preliminaries and Problem Definition

A. Preliminaries

We consider a database $\mathcal{D}$ of spatial objects. Each object $o \in \mathcal{D}$ is a pair $(o.loc, o.doc)$, where $o.loc$ is a point location and $o.doc$ is a set of generic keywords that describe the object. A spatial keyword top-k query q retrieves k top ranked objects from $\mathcal{D}$ according to a ranking function that takes into consideration both spatial distance and textual similarity. For broad applicability, we consider a widely used ranking function [10]:

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

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

A spatial keyword top-k query q is a 4-tuple (loc, doc, k, α), where $q.loc$ is a query point location, $q.doc$ is a set of query keywords, $q.k$ is the number of objects to retrieve, and $q.\alpha$ denotes the user preference. The distance $SDist(o,q)$ is calculated as the (normalized) Euclidean distance between $p.loc$ and $q.loc$. The textual similarity $TSim(o,q)$ can be computed using an information retrieval model. Without loss of generality, we adopt the widely-used Jaccard similarity model [24]:[1]

$$TSim(o,q) = \frac{|o.doc \cap q.doc|}{|o.doc \cup q.doc|} \quad (2)$$

In the ranking function, the higher the score computed by Eqn. (1), the higher the ranking of the corresponding object. We define the rank of an object o as follows:

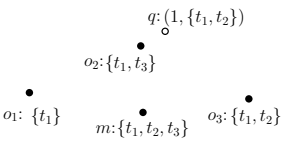
$$R(o,q) = |\{o' \in \mathcal{D} | ST(o',q) > ST(o,q)\}| + 1 \quad (3)$$

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 a set $\mathcal{R}$ of k objects from $\mathcal{D}$, where $\forall o \in \mathcal{R}\ (\forall o' \in \mathcal{D} - \mathcal{R}\ (ST(o,q) \geq ST(o',q)))$, or in terms of object rank, $\forall o \in \mathcal{R}\ (\forall o' \in \mathcal{D} - \mathcal{R}\ (R(o,q) \leq R(o',q)))$.

B. Why-Not Spatial Keyword Query

It is often difficult for users to find the keywords that best describe their query intention. Thus, identifying the proper query keyword set is arguably the key challenge in issuing a spatial keyword top-k query. After a user issues a query $q = (loc, doc_0, k_0, \alpha)$ and receives the result, the user may observe that one or more objects that were expected to be in the result are missing. The user may then pose a *why-not* question with a set of missing objects $M = \{m_1, m_2, ..., m_j\}$, asking the system for a refined query $q' = (loc, doc', k', \alpha)$ the result of which contains the missing objects. Since it is possible that no modified set of keywords can revive the missing objects, we also consider the enlargement of k. Different modifications of these two parameters may yield many qualified queries that retrieve the missing objects, and we prefer the refined query that minimally modifies the original query. Specifically, we adopt a penalty model [8], [13], [15] that associates a penalty with a refined query. It is defined as the weighted sum of the modifications of the two parameters, *i.e.*, Δk and Δdoc. The penalty (cost) of a query q' that refines an original query q is defined as follows:



(a)

	$1 - SDist(o,q)$	$TSim(o,q)$	$ST(o,q)$
m	0.5	0.66	0.58
o_1	0.2	0.5	0.35
o_2	0.9	0.33	0.615
o_3	0.4	1	0.7

(b)

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

$$Penalty(q,q') = \lambda \cdot \frac{\Delta k}{R(M,q) - k_0} + (1-\lambda) \cdot \frac{\Delta doc}{|doc_0 \cup M.doc|} \quad (4)$$

Here, λ is a user preference on the modification of $q.k$ versus $q.doc$ and $R(M,q) = max_{m_i \in M} R(m_i, q)$. Next, $\Delta k = max(0, k' - k_0)$ since for a refined query q', if $R(M,q') > k_0$, k' must be no smaller than $R(M,q')$ to revive the missing objects; otherwise, k' can remain at k_0. As in other studies [8], [15], we normalize Δk by $R(M,q) - k_0$, as a basic refined query is to keep the original query keywords and enlarge k_0 to $R(M,q)$; for other refined queries that modify the query keywords to achieve a lower penalty than that of this basic one, Δk must not exceed $R(M,q) - k_0$. Using the principle of edit distance, the modification of query keywords Δdoc is quantified as the minimum count, denoted as $ED(doc_0, doc')$, of edit operations needed to transform doc_0 to doc'. For simplicity, we consider two edit operations: insertion and deletion. Similarly, we normalize $ED(doc_0, doc')$ by the maximum possible number of edit operations needed to modify doc_0 into a doc' that yields a query that retrieves all objects in M. This quantity is estimated as $|doc_0 \cup M.doc|$, where $M.doc = \bigcup_{i=1}^{j} m_i.doc$. In other words, we just consider the keywords in $M.doc$, as adding a keyword not in $M.doc$ would make the set of query keywords less relevant to the user's query intention, *i.e.*, less relevant to the missing objects.

We now define the keyword-adapted why-not spatial keyword top-k queries as follows:

Definition 2: **Keyword-Adapted Why-Not Spatial Keyword Top-k Query**. Given a set $\mathcal{D}$ of spatial objects, a missing object set $M \subset \mathcal{D}$, an original spatial keyword query $q = (loc, doc_0, k_0, \alpha)$, the *keyword-adapted why-not spatial keyword top-k query* returns the refined query $q' = (loc, doc', k', \alpha)$, with the lowest penalty according to Eqn. (4) and the result of which contains all objects in M.

TABLE I. An Example of Refined Queries

Refined Query	Δk	Δdoc	Penalty
$q_1 = (3, \{t_1, t_2\})$	1	0	0.5
$q_2 = (1, \{t_2, t_3\})$	0	0.66	**0.33**
$q_3 = (2, \{t_1, t_3\})$	0.5	0.66	0.58
$q_4 = (2, \{t_1, t_2, t_3\})$	0.5	0.33	0.415

[1] The algorithm developed in Section IV can support other similarity models by using the particular spatial keyword top-k query algorithms associated with these models; the KcR-tree based algorithm proposed in Section V can also be extended to support other models, *e.g.,* the Dice coefficient and the Cosine similarity.

Example 3: Fig. 1 illustrates the keyword-adapted why-not query. Figure 1(a) shows the locations and keywords of four objects and the initial query q. Figure 1(b) lists the values of $1 - SDist(o, q)$, $TSim(o, q)$ as well as the ranking score $ST(o, q)$ of each object with respect to (w.r.t.) the initial query that has $doc_0 = \{t_1, t_2\}$, $k_0 = 1$, and $\alpha = 0.5$. According to the $ST(o, q)$ values, we can see that object m has rank 3 under the initial query, so it is not in the result. Table I lists several refined queries that retrieve m together with their penalties, where $\lambda = 0.5$. We omit loc and α in the refined queries as they stay unchanged. In this setting, $R(m, q) - k_0 = 2$ and $|doc_0 \cup m.doc| = 3$. Query q_1 keeps the initial query keywords and enlarges k_0 to $R(m, q)$. So the total penalty is $0.5 \cdot \frac{3-1}{2} + 0.5 \cdot \frac{0}{3} = 0.5$. Similarly, q_2 keeps k_0 and changes the query keywords to $\{t_2, t_3\}$, which causes a smaller penalty. Query q_2 is the best refined query.

IV. Basic Why-not Answering Algorithm

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

A. Problem Analysis

Recall that we consider refining the query keywords and k to achieve the inclusion of missing objects. Only refined pairs (doc', k') that satisfy Lemma 1 are candidates for the best refined query.

Lemma 1: Given a set M of missing objects, a pair of refined query keywords and a result cardinality (doc', k') can result in the best refined query if and only if (i) $k' = R(M, q')$, when $R(M, q') \geq k_0$; or (ii) $R(M, q') \leq k' \leq k_0$, when $R(M, q') < k_0$, where $R(M, q') = max_{m_i \in M} R(m_i, q')$.

The proof is straightforward and hence omitted. Lemma 1 implies that given a refined keyword set doc', we may set $k' = R(M, q')$ to get the minimum penalty. This observation inspires a basic solution to the keyword-adapted why-not query. In the following, we first consider a single missing object m. In Section VI, we show how the algorithms can be adapted to handle multiple missing objects.

B. Basic Algorithm

The basic algorithm works as follows. First, we determine the rank $R(m, q)$ of the missing object m w.r.t. the initial spatial keyword query by processing the query until object m appears. Then we enumerate all possible query keyword sets and invoke an existing spatial keyword top-k query algorithm (*e.g.,* [10]) for each such set to determine the ranking of object m for each keyword set. Finally, we return the keyword set with the minimum penalty according to Eqn. (4).

To support spatial-keyword top-k query processing with the Jaccard similarity model, similar to related work [10], [26], we employ a hybrid index that estimates bounds on spatial distance and textual similarity at the same time. This index, called the SetR-tree, is a variant of the IR-tree [20]. A leaf node of the SetR-tree stores a number of entries of the form (o, mbr, pks), where o represents an object, mbr is the minimum bounding rectangle (MBR) of the object, and pks is a pointer to the keyword set of o. A non-leaf node stores a number of entries of the form (pc, mbr, pku, pki), where pc is a child pointer, mbr is the MBR of the child node, pku is a pointer to the union of the keyword sets of all the objects indexed in the subtree rooted at this node, and pki is a pointer to the intersection of the keyword sets. These two sets are stored sequentially on disk to reduce the number of disk seeks. Fig. 2 shows an example SetR-tree, where the keyword sets associated with each leaf and non-leaf node are shown. For instance, consider the non-leaf node R_3; its union (resp., intersection) set is the union (resp., intersection) of keyword sets associated with leaf nodes R_1 and R_2.

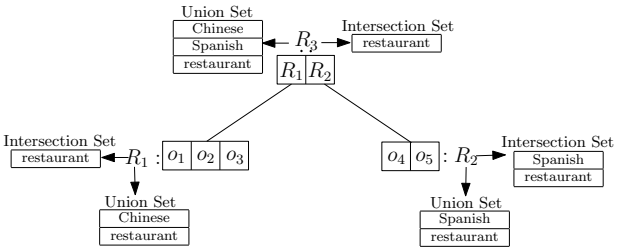


Fig. 2. An Example SetR-tree

Theorem 1: Consider a query $q = (loc, doc, k, \alpha)$ and a SetR-tree node N that indexes a set $\mathcal{S}$ of objects. Let $MDist(q.loc, N.MBR)$ denote the minimum distance between $q.loc$ and N's MBR and let $N_\cup$ and $N_\cap$ denote the union and intersection keyword sets of N, respectively. It holds that:

$$\forall o \in \mathcal{S}\ (ST(o, q) \leq \alpha \cdot (1 - MDist(q.loc, N.MBR)) + (1 - \alpha) \cdot \frac{|N_\cup \cap q.doc|}{|N_\cap \cup q.doc|}) \quad (5)$$

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

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

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

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

Recall that $ST(o, q) = \alpha \cdot (1 - SDist(o.loc, q.loc)) + (1 - \alpha) \cdot TSim(o.doc, q.doc)$, where neither of the two terms exceeds those of Eqn. (5). Thus, Theorem 1 holds. ■

Theorem 1 enables us to estimate the bounds for spatial distance and textual similarity using the SetR-tree, which further provides the bound of the ranking score of each object indexed by a SetR-tree. This facilitates the search by a SetR-tree to find the top-k most relevant objects. The spatial keyword query processing algorithm using the SetR-tree is similar to that of the IR-tree [20]. Interested readers may refer to [20] for details.

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

C. Optimizations

The above algorithm essentially executes a spatial keyword query for each candidate keyword set doc' to determine the ranking of the missing object for the set. The doc' sets are obtained by adding keywords in $m.doc$ to doc_0 and by deleting keywords originally in doc_0. This yields $2^{|doc_0 \cup m.doc|}$ candidate keyword sets. For each such keyword set, a spatial keyword query is processed until the missing object is retrieved. We now present techniques that accelerate this process.

1) Early Stop: In the basic algorithm, for each doc', we need to invoke a spatial keyword query to determine the ranking of the missing object, which means that the query can stop only when the missing object m appears in the result. However, for a keyword set doc' that is not very relevant to m, the number of objects that rank higher than m may be very large. Fortunately, such extensive computations can be avoided. We have seen that a basic refined query is to just enlarge k in the original query to $R(m, q)$ without modifying the query keyword set. This refined query has penalty λ according to Eqn. (4). Our approach is to maintain a currently best refined query together with its penalty p_c. The penalty of a refined query is the sum of two terms: the penalty for modifying k_0 and the penalty for modifying doc_0. Given a keyword set doc', we can compute Δdoc accordingly. The query q' generated from doc' can be the best refined query if and only if:

$$\lambda \cdot \frac{max(0, R(m, q') - k_0)}{R(m, q) - k_0} + (1 - \lambda) \cdot \frac{\Delta doc}{|doc_0 \cup m.doc|} \leq p_c,$$

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

$$R_L(m, q') = \lfloor k_0 + \frac{p_c - (1 - \lambda) \cdot \frac{\Delta doc}{|doc_0 \cup m.doc|}}{\lambda} \cdot (R(m, q) - k_0) \rfloor \quad (6)$$

Eqn. (6) implies that we can stop processing the spatial keyword query corresponding to a keyword set doc' as soon as we retrieve more than $R_L(m, q')$ objects. When a new query q' is found that has a smaller penalty than the currently smallest penalty, p_c is updated and query q' is recorded as the currently best refined query.

Example 4: Consider a top-5 query, where the ranking $R(m, q)$ of the missing object under the initial query is 10 and $\lambda = 0.5$. Assume that the currently best refined query q_c has penalty $p_c = 0.5$. Now we are going to process a candidate keyword set doc' w.r.t. which the ranking of the missing object is 100. Suppose $\frac{\Delta doc}{|doc_0 \cup m.doc|} = 0.4$. Then according to Eqn. (6), we can calculate $R_L(m, q') = \lfloor 5 + \frac{0.5 - (1-0.5) \cdot 0.4}{0.5} \cdot (10 - 5) \rfloor = 8$, which means we can stop the processing of the query generated by doc' as soon as we retrieve 8 objects.

2) Enumeration Order: The order in which we consider the candidate keyword sets plays an important role in the basic algorithm. First, p_c plays a key role in the pruning, which further determines $R_L(m, q')$ for an incoming doc'. Finding a small p_c early can speed up the pruning and improve the efficiency of the algorithm. Second, the dominating objects of the missing object in the previous queries have high chances to keep dominating m if the next keyword set is similar to the previous ones. Thus, we consider the keyword sets in a particular order. We should first consider the keyword sets who are more likely to be the final best refined query. We estimate this *"more likely"* based on two conjectures: keyword sets with a smaller edit distance to the original keyword set are *"more likely"* since they incur lower penalty; and adding a keyword that is more particular to a missing object will make the query more related to m. We define the particularity of a keyword t to an object o using the IDF (inverse document frequency) weight as follows:

$$Parti(o, t) = \begin{cases} -log \frac{|D| - n_t + 0.5}{n_t + 0.5} & t \notin o.doc \\ log \frac{|\mathcal{D}| - n_t + 0.5}{n_t + 0.5} & t \in o.doc \end{cases}, \quad (7)$$

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

3) Keyword Set Filtering: Since similar keyword sets have similar textual similarities to an object, the rankings of objects according to the scoring function (Eqn. (1)) may be similar for similar query keyword sets. For instance, if we have processed a spatial keyword query for a keyword set doc_0 and have obtained the set C of objects that rank higher than missing object m (*i.e.*, dominators of m) then for the keyword set doc' that is obtained by slightly modifying doc_0, *e.g.,* removing a keyword from doc_0, the textual similarity and ranking for each object in C will change very little. Thus, the objects in C will stand a good chance to keep ranking higher than m. Therefore, we propose to cache the dominators of the missing object retrieved from the previously processed query. Before generating and processing a spatial keyword top-k query for a new keyword set doc', we test how many cached dominators still rank higher than the missing object. If this number exceeds the lower bound of the ranking we derived for doc' using Eqn. (6), doc' can be pruned safely without further processing.

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

Algorithm 1 summarizes the basic algorithm along with the optimizations.

V. Bound-and-Prune Algorithm

While the optimizations make the basic algorithm more efficient, it still needs to invoke a spatial keyword query for each candidate keyword set one by one, where each time multiple SetR-tree nodes need to be loaded into memory and accessed. As I/O operations are much more costly than in-memory operations, loading the index nodes and data objects for each keyword set is wasteful. Also, the intersection/union keyword

Algorithm 1 Basic Algorithm for Answering Why-not Questions

INPUT: SetR-tree $\mathcal{T}$, original query $q = (loc, doc_0, k_0, \alpha)$, missing object m
OUTPUT: Best refined query $q' = (loc, doc', k', \alpha)$

determine $R(m, q)$
$doc' \leftarrow doc_0, k' \leftarrow R(m, q), p_c \leftarrow \lambda$ //initialize the best refined query and the penalty threshold using the basic refined query
$S \leftarrow \emptyset$ // a candidate query keyword set
repeat
  $S \leftarrow$ ***NextKeywordSet()*** // find the next query keyword set according to the enumeration order
  if $(1 - \lambda) \cdot \frac{\Delta doc}{|doc_0 \cup m.doc|} \geq p_c$
    break
  compute R_L for S according to Eqn. (6)
  $t \leftarrow 0$
  for each object o in C
    if $ST(o, q_S) > ST(m, q_S)$ **then** $t++$ //q_S is the spatial keyword query generated by the keyword set S
  if $t \geq R_L$
    continue
  determine $R(m, S)$ by processing a spatial keyword query // in query processing, if the number of retrieved objects exceeds R_L, the processing terminates and the algorithm continues to the next keyword set
  compute the penalty p for S according to Eqn. (4)
  if $p < p_c$ **then**
    $p_c \leftarrow p$, $doc' \leftarrow S$, $k' \leftarrow R(m, S)$
until $S = \emptyset$
return $q' = (loc, doc', k', \alpha)$

sets in an upper node of the SetR-tree may be empty/large if the underlying objects are from different categories, which would further slow down the processing of a spatial keyword query. To address these problems, we present an index-based bound-and-prune algorithm that finds the best sample out of a number of keyword sets and prunes the others in just one index access. As the index nodes and data objects are loaded just once for a set of keyword sets, substantial I/O can be saved.

A. Preliminaries: KcR-Tree

We adopt the KcR-tree (*Keyword count R-tree*) [22] as the index structure. Essentially, a KcR-tree is an R-tree that integrates the textual information of the indexed objects into each tree node.

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

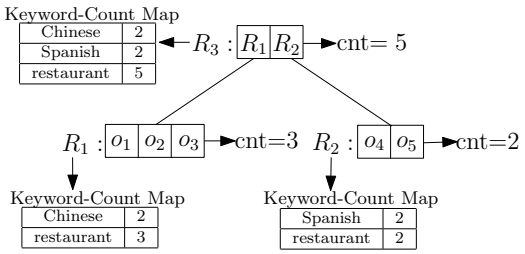


Fig. 3. An Example KcR-tree

B. Properties of KcR-tree

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

Theorem 2: Given a KcR-tree node N, an initial query q, a missing object m, and a refined query keyword set S, if an object o indexed in N ranks higher than m under the new query, the following inequality must hold:

$$TSim(o, S) > \frac{\alpha}{1-\alpha} \cdot (MinDist(N, q) - SDist(m, q)) + TSim(m, S), \quad (8)$$

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

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

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

which can be rewritten as:

$$TSim(o, S) > \frac{\alpha}{1-\alpha} \cdot (SDist(o, q) - SDist(m, q)) + TSim(m, S)$$

Since object o is enclosed by node N's minimum bounding rectangle, the following is true:

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

The theorem follows by combining the two inequalities. ■

Theorem 2 provides a lower bound of the textual similarity for an object o in N that ranks higher than the missing object w.r.t. query keyword set S. We denote the lower bound as $TSim(N, o, S)_L$. Next, we introduce another important metric, the pseudo textual similarity between a KcR-tree node N and a keyword set S, which will be used in the estimation of the ranking bounds.

Definition 3: The pseudo textual similarity between a KcR-tree node N and a keyword set S is defined as follows:

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

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

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

Theorem 3: Given a KcR-tree node N, an initial query q, a missing object m, and a refined keyword set S, all the objects indexed in N are dominators of m if and only if:

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

Proof: Based on Theorem 2, to ensure each object indexed in N is a dominator of m, it holds that $\forall o \in N$ ($TSim(o, S) > TSim(N, o, S)_L$). According to the definition of textual similarity, $TSim(o, S) = \frac{|o.doc \cap S|}{|o.doc \cup S|} \geq TSim(N, o, S)_L$. Therefore, for each object $o \in N$, the inequality $|o.doc \cap S| \geq |o.doc \cup S| \cdot TSim(N, o, S)_L$ holds, from which we can further deduce that:

$$\sum_{o \in N} |o.doc \cap S| \geq \sum_{o \in N} |o.doc| \cdot TSim(N, o, S)_L$$

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

Theorem 3 implies that, given a KcR-tree node N, if the inequality $TSim^{\sim}(N, S) \geq TSim(N, o, S)_L$ does not hold, at least one object in N ranks lower than the missing object m. This suggests a possibility of estimating $MaxDom(N, S, m)$ and allows us to develop Algorithm 2 to derive it. We first assume that all objects in N dominate the missing object m, *i.e.,* $MaxDom(N, S, m) = N.cnt$ (Line 4). We then iteratively virtually prune objects from N until Theorem 3 holds for the consequent virtual node and return the number of remaining objects as the upper bound on m's dominators in N (Lines 5–14). As we do not know the keyword sets of each object in N, to find the upper bound $MaxDom(N, S, m)$, we assume all the virtually pruned objects are irrelevant to the set of query keywords. We associate the irrelevant (resp., relevant) keywords with the pruned (resp., remaining) objects as many as possible. To do so, we divide the keywords in $N.doc$ into two categories, *i.e.,* $N.doc - S$ and $N.doc \cap S$. The keywords in $N.doc \cap S$, which are relevant to the query keywords, are kept for the remaining objects, except we have to associate them with the pruned ones, *i.e.*, $N.count(t) > ans$ (Line 11). The keywords in $N.doc - S$, which are irrelevant to the query keywords, are pruned as long as we can associate them with the pruned object, *i.e.,* the keyword set $T_{N-S} \leftarrow \{t \mid t \in N.doc - S \wedge N.count(t) \geq (N.cnt - ans)\}$ (Line 12). We omit the details of the estimation of $MinDom(N, S, m)$ as it is done similarly.

Example 5: Consider a KcR-tree node N, whose keyword-count map is $\{(t_1, 8), (t_2, 3), (t_3, 7), (t_4, 2), (t_5, 1)\}$ and $N.cnt = 8$. Assume the query keyword set $S = \{t_3, t_4\}$ and that we have computed $TSim(N, S, o)_L = 0.395$. In this setting, $S \cap N = \{t_3, t_4\}$ and $N - S = \{t_1, t_2, t_5\}$. We start by assuming all the objects indexed in N rank higher than the missing object m w.r.t. S, i.e.*, $MaxDom(N, S, m) = 8$. We try to verify our assumption using Theorem 3. We find that $TSim^{\sim}(N, S) = \frac{9}{2 \cdot 8 + 12} = \frac{9}{28} < TSim(N, S, o)_L$. Our assumption fails. Then we modify our assumption on $MaxDom(N, S, m)$ and virtually prune an object o from the node by associating as many irrelevant keywords as possible with o, which leads to a virtual KcR-tree node N' that indexes 7 objects and has keyword-count map $\{(t_1, 7), (t_2, 2), (t_3, 7), (t_4, 2), (t_5, 0)\}$. Again, we test the assumption and find $TSim^{\sim}(N, S) = \frac{9}{23}$, which is still less than 0.395. Hence, we prune one more object from N'. However, as $N'.count(t_5) = 0$, we cannot associate t_5 with the pruned objects; on the other hand, as $N'.count(t_3) = N'.cnt$, we have to associate it with the pruned object. After this iteration, the pseudo textual similarity between the consequent node and S is 0.4, which exceeds $TSim(N, S, o)_L$. Thus, the function terminates and returns 6 as $MaxDom(N, S, m)$.*

Algorithm 2 MaxDom(N,S,m)

INPUT: A KcR-tree Node N, a keyword set S, the missing object m
OUTPUT: $MaxDom(N, S, m)$

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

C. Optimized Bound-and-Prune Algorithm

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

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

refined query with the query generated by S if $\hat{pn}(S)$ is less than its penalty.

The optimized bound-and-prune algorithm determines the best refined query keyword set among a set of candidates in just one traversal of the KcR-tree. It estimates the missing object's ranking as well as the penalty w.r.t. each candidate keyword set S before actually unfolding a KcR-tree node. The algorithm is detailed in Algorithm 3. It takes as input a currently known best refined query q' and a set of candidate keyword sets CK, and it returns the best refined query among the input q' and the queries generated by each query keyword set in CK. The currently best refined query is initialized to the basic refined query that just sets the number of retrieved objects to $R(m, q)$ and keeps the initial query keywords. Let $\hat{D}(N, S)$ and $\check{D}(N, S)$ denote the upper and lower bounds of the number of objects in a KcR-tree node N that rank higher than the missing object m, which can be computed by the two functions $MaxDom(N, S, m)$ and $MinDom(N, S, m)$, respectively. First, we estimate the upper bound and lower bound of the ranking of the missing object m w.r.t. each candidate keyword set S as $\check{D}(\mathcal{T}.root, S)+1$ and $\hat{D}(\mathcal{T}.root, S)+1$ (Lines 2–6). Then we insert the root $\mathcal{T}$ into the queue $\mathcal{Q}$ (Line 7) and start to traverse the KcR-tree. In each iteration, we dequeue a KcR-tree node N from $\mathcal{Q}$ (Line 9). By unfolding N, we obtain more detailed spatial and textual information. We next access each child c of N and compute $\hat{D}(c, S)$ and $\check{D}(c, S)$ by calling the $MaxDom(c, S, m)$ and $MinDom(c, S, m)$ functions (Lines 14–15). By doing so, we get tighter upper and lower bounds of objects in N that rank higher than the missing object, *i.e.*, $\hat{D}'(N, S) = \sum_{c \in N} \hat{D}(c, S)$ and $\check{D}'(N, S) = \sum_{c \in N} \check{D}(c, S)$ (Lines 16–17). Obviously, the difference between $\hat{D}(N, S)$ and $\hat{D}'(N, S)$ is the number of objects that are considered wrongly to rank higher than m when we just access N. Thus, we can re-estimate the upper bound of m's ranking w.r.t S more accurately by subtracting this difference (Line 18). The tightened lower bound of the ranking of m under S can be computed similarly (Line 19). With the tighter bounds on $m's$ ranking w.r.t S, we can compute its penalty upper bound and lower bound according to the penalty function Eqn. (4) (Line 20). We update the currently known best refined query with the query generated by S if the penalty upper bound $\hat{pn}(S)$ of S is smaller than p_c (Lines 21–23). Otherwise, we prune a candidate S if $\check{pn}(S) > p_c$ (Lines 25–26). Also, we prune the nodes that cannot further tighten the bounds (Lines 29–30). We return the best refined query after $\mathcal{Q}$ or CK becomes empty (Lines 27–28, Line 33).

Algorithm 3 KcR-tree Based Algorithm for Answering Why-not Questions

INPUT: KcR-tree $\mathcal{T}$, current best refined query $q' = (loc, doc', k', \alpha)$, candidate keyword sets CK, missing object m, current best penalty p_c
OUTPUT: Best refined query $q' = (loc, doc', k', \alpha)$ among keyword sets in CK and its penalty p_c

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

D. Strategic Use of Algorithm 3

Algorithm 3 finds the best refined query among a set of candidate keyword sets. One straightforward way of using Algorithm 3 to answer a why-not query is to first generate all the candidate keyword sets and input them to the algorithm. However, the performance of Algorithm 3 is dominated by the execution time of the $MaxDom$ and $MinDom$ functions, which is proportional to the number of candidate keyword sets. While the straightforward way of using the algorithm may work well for a small number of candidate keyword sets, we proceed to present a strategy to divide all the candidates into subsets according to their penalty, *i.e.*, the edit distance, from the initial query keyword set. This helps speed up the process and triggers early stop that avoids enumerating all candidates. We access the subsets in ascending order of their penalty, and for each such subset, we invoke Algorithm 3 to determine the best refined query. The process stops when the penalty of the known best refined query is no larger than that of the next retrieved subset.

The details are shown in Algorithm 4. The first step is to determine the ranking of the missing object under the initial query, *i.e.*, $R(m, q)$ (Line 1). This can be done by slightly modifying the underlying spatial-keyword top-k algorithm by changing the stop condition from retrieving top-k objects to retrieving the missing object m. Then we initialize the currently best refined query to be the basic one (Line 2). Afterwards, we iteratively find the subset of query keywords in ascending order of their edit distance to the initial query keywords and invoke Algorithm 3 until the next subset's

penalty in the textual dimension is no less than that of the known best refined query (Lines 3–7).

Algorithm 4 Answering Why-not Query

INPUT: KcR-tree $\mathcal{T}$, original query $q = (loc, doc_0, k_0, \alpha)$, missing object m
OUTPUT: Best refined query $q' = (loc, doc', k', \alpha)$

1: determine $R(m, q)$
2: $doc' \leftarrow doc_0, k' \leftarrow R(m, q), p_c \leftarrow \lambda$ // initialize the best refined query and the penalty threshold with the very basic refined query
3: **for** k from 1 to $|doc_0 \cup m.doc|$ **do**
4: **if** $(1 - \lambda) \cdot \frac{k}{|doc_0 \cup m.doc|} \geq p_c$ **then**
5: **break**
6: $CK \leftarrow$ ***NextKeywordSets()*** // find the next set of keyword sets that has k edit distance from doc_0
7: invoke Algorithm 3 using $(\mathcal{T}, q', CK, p_c, m)$ to determine the currently best refined query q' and its penalty p_c
8: **return** $q' = (loc, doc', k', \alpha)$

VI. Multiple Missing Objects and Approximate Algorithm

A. Multiple Missing Objects

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

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

To achieve the inclusion of all missing objects, we slightly modify the algorithms. First, we use $R(M, q)$ instead of $R(m, q)$ when estimating the penalty (Eqn. (4)). In the basic algorithm, the largest modification is to stop a generated spatial keyword query when all missing objects are retrieved. Similarly, in the KcR-tree based algorithm, for each candidate keyword set, $MaxDom(\mathcal{T}, S, M)$ and $MinDom(\mathcal{T}, S, M)$ are estimated as $max_{m_i \in M} MaxDom(\mathcal{T}, S, m_i)$ and $min_{m_i \in M} MinDom(\mathcal{T}, S, m_i)$, respectively. The estimation of the missing objects' rankings ($\hat{R}(S)$ and $\check{R}(S)$) and the penalty ($\hat{pn}(S)$ and $\check{pn}(S)$) w.r.t. each candidate keyword set S is changed accordingly. All the optimization techniques can be adapted similarly to support queries with multiple missing objects.

B. Approximate Algorithm

So far, we have focused on finding the exact solution with the least penalty among all candidate keyword sets. The number of candidate keyword sets grows fast when $|doc_0 \cup M.doc|$ increases. Although several optimizations and an index-based algorithm are proposed for the why-not question, the exact algorithm could still be costly when the number of the query keywords is huge. In such cases, we can trade solution quality for execution time. Instead of taking into consideration of all the candidate keyword sets, we apply the algorithms only to a sample of all sets to find an approximate solution. As with typical sampling-based methods [13], [15], two key issues need to be addressed: the sample size and how to obtain high quality keyword sets in the sample.

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

To sample high quality keyword sets, based on the analysis of the *enumeration order* that we discussed in Section IV-C2, we greedily choose the first T keyword sets with the highest total particularity w.r.t. the missing objects due to editing the initial query keyword set.

VII. Empirical Study

The ensuing experimental study primarily considers three methods: the basic algorithm developed in Section IV-B (referred to as ***BS***), the basic algorithm with the optimizations from Section IV-C (referred to as ***AdvancedBS***), and the KcR-tree approach with the optimizations developed in Section V (referred to as ***KcRBased***). We also implement the approximate algorithm and evaluate its performance and solution quality.

A. Experimental Setup

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

2) Datasets: We use two real datasets, EURO and GN, in the experiments. EURO is a dataset of points of interest like ATMs, hotels, and stores in Europe (www.allstays.com); and GN is obtained from the US Board on Geographic Names (geonames.usgs.gov) and contains a set of geographic objects. Both of them are commonly used in spatial keyword related research [4], [5], [8], [23], [31]. Each dataset contains a number of objects represented by a spatial location and a set of keywords. More details about the datasets are provided in Table II.

TABLE II. DATASET INFORMATION

Dataset	EURO	GN
Total # of objects	162,033	1,868,821
Total # of distinct words	35,315	222,407

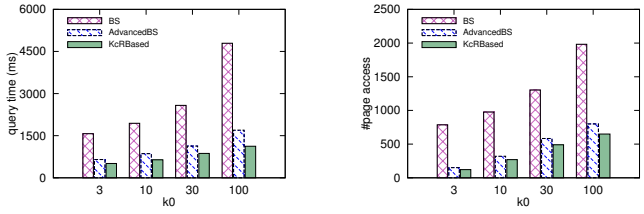
3) Parameters: We evaluate the performance of our algorithms by varying different parameters. The parameters together with their default values (in bold face) are shown in Table III. Unless specified otherwise, the experiments are conducted using the dataset EURO, and we set the missing object as the one ranked at $5 \cdot k_0 + 1$ in the initial query.

TABLE III. PARAMETER SETTING

Parameter	Setting
k_0	3, **10**, 30, 100
# of keywords	2, **4**, 6, 8
α	0.1, 0.3, **0.5**, 0.7, 0.9
$R(m, q)$	31, **51**, 101, 151, 201
λ	0.1, 0.3, **0.5**, 0.7, 0.9
# of missing objects	**1**, 2, 3, 4

B. Experimental Result

1) Varying k_0: We first vary parameter k_0 in the initial query to observe its effect on the performance of the algorithms. The ranking of the missing object varies along with k_0, *i.e.*, $R(m, q) = 5 \cdot k_0 + 1$. For instance, when the initial query is varied from a top-3 to a top-10 query, corresponding why-not queries are posed to identify keyword sets that retrieve objects that rank from 16 to 51. Fig. 4 shows the results. Recall that in the basic algorithm, a spatial keyword query is executed for each candidate keyword set until the missing object is retrieved. Since the ranking of the missing object drops as k_0 increases, the time for executing a spatial keyword query increases. Thus, the basic algorithm is quite sensitive to changes in k_0. On the other hand, thanks to the optimization techniques developed, ***AdvancedBS*** and ***KcRBased*** scale much better when k_0 increases, and ***KcRBased*** achieves the best performance. For example, when $k_0 = 100$, *KcRBased* runs almost 5 times faster than ***BS*** and reduces the I/O by more than 70%.

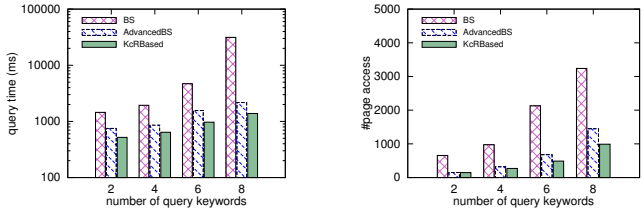


(a) Query Time (b) Page Access

Fig. 4. Varying k_0

2) Varying the number of initial query keywords: This set of experiments evaluates the effect of varying the number of query keywords in the initial query. The results are shown in Fig. 5. Intuitively, the number of keywords influences the performance of the algorithms in two aspects. First, more query keywords mean that more time will be consumed to compute the textual similarity between tree nodes/objects and query keywords; second, the candidate query keyword sets may grow exponentially when the number of initial query keywords increases.

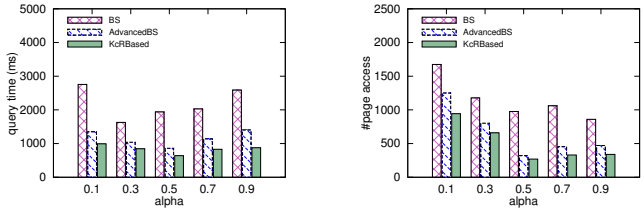
The number of candidate keyword sets is a dominant factor. As the basic algorithm needs to process a spatial keyword query to determine the rank of the missing object under each candidate keyword set, the query time of ***BS*** increases dramatically when the number of initial query keywords increases.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



(a) Query Time(log-scale) (b) Page Access

Fig. 5. Varying the number of initial query keywords

3) Varying α: In this set of experiments, we study the effect of parameter α. Fig. 6 plots the results. According to the ranking function in the spatial keyword top-k query, *i.e.*, Eqn. (1), a smaller α means a higher weight to the textual similarity, which reduces the importance of spatial distance. As a result, the pruning ability of the R-tree based indexes decreases, and more tree nodes may need to be accessed. That is the main reason why a smaller α causes more I/O in all tested algorithms. On the other hand, as a large (resp., small) α gives higher weight to the spatial (resp., textual) dimension in the ranking function, this reduces the pruning capability in the other dimension. This may be the reason why a medium α has the least execution time.

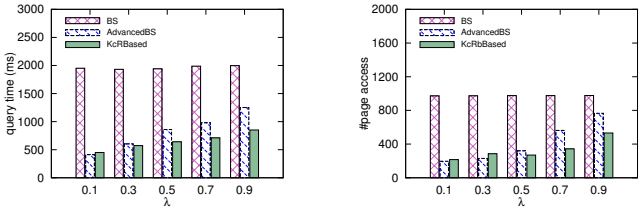


(a) Query Time (b) Page Access

Fig. 6. Varying α

4) Varying λ: Next we investigate the effect of parameter λ in the penalty function, which allows users to indicate their preferences on modifying the query keywords versus modifying k. As shown in Fig. 7, the basic algorithm is almost unaffected by λ. The reason is that in ***BS***, λ is only used to compute the penalty of a candidate keyword set after the generated spatial keyword query determines the rank of the missing object; the computation is exactly the same for different λ values. However, in both ***AdvancedBS*** and ***KcRBased***, the currently best refined query and its penalty are maintained for further pruning, which is initialized using the basic refined query that keeps the query keywords and sets k_0 to $R(m, q)$ to include the missing object. According to Eqn. (4), the penalty of the basic refined query is λ. A smaller λ leads to a smaller initially seen penalty in ***AdvancedBS*** and ***KcRBased***, which can improve the pruning ability. Therefore, the query time of ***AdvancedBS*** and ***KcRBased*** increases with λ. However, ***KcRBased*** is more stable than ***AdvancedBS***.

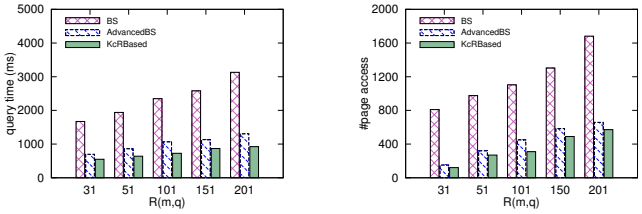
5) Varying the rank of the missing object: In this set of experiments, we study the performance when the initial rank



(a) Query Time (b) Page Access

Fig. 7. Varying λ

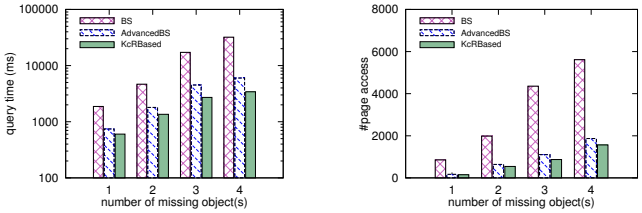
of the missing object is varied. Since the initial query is a spatial keyword top-10 query, we ask five different why-not questions with the missing object ranked 31, 51, 101, 151, and 201. As shown in Fig. 8, the performance of ***BS*** is much more sensitive to changes in the rank of the missing object, whereas ***AdvancedBS*** and ***KcRBased*** are affected only slightly. The reason is the same as when varying k_0. In fact, the results of this set of experiments and those of varying k_0, *i.e.*, Fig. 8 and Fig. 4, are quite similar. This implies that the performance of the algorithms is affected significantly by the initial rank of the missing object and has little to do with k_0.



(a) Query Time (b) Page Access

Fig. 8. Varying the missing object's initial ranking

6) Varying the number of missing objects: We also study performance when the number of the missing objects changes. In this set of experiments, the initial query is a top-10 spatial keyword query with 4 query keywords. The missing objects are randomly selected from the objects ranked between 11 and 51 w.r.t. the initial query. Fig. 9 plots the results. We can observe that the number of missing objects has a remarkable effect on the performance of the algorithms. The reason is that for multiple missing objects, we need to consider the union of all the missing objects' keywords as the search space, which means that the number of the candidate keyword sets may increase dramatically along with the increasing number of missing objects. We can also see that ***AdvancedBS*** and ***KcRBased*** scale much better than ***BS*** when the number of missing objects grows.



(a) Query Time(log-scale) (b) Page Access

Fig. 9. Varying the number of missing objects

7) Varying the number of threads: In Fig. 10, we further evaluate the performance of the algorithms when the spatial keyword queries are processed in parallel. The ***KcRBased*** algorithm is parallelized by dividing the set of candidate keyword sets into several smaller sets and running the algorithm on each set while the currently known best refined query and its penalty are synchronized for pruning and early termination. The performance of the algorithms can be accelerated significantly by using up to 8 threads.

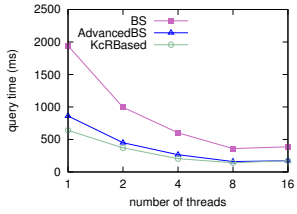


Fig. 10. Varying the number of threads

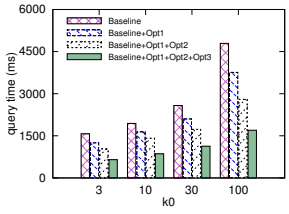
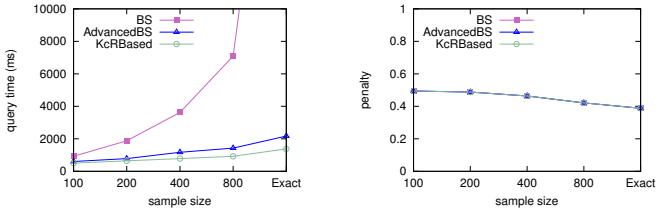


Fig. 11. Effects of different optimizations

8) Pruning abilities of optimizations: We show the query performance of different optimizations in Fig. 11, where Opt1, Opt2, and Opt3 represent the strategies of early stopping, considering the enumeration order, and keyword set filtering, respectively. Each optimization reduces the query time. In particular, keyword set filtering is the most effective, as it prunes candidate keyword sets before even processing the corresponding spatial keyword queries.



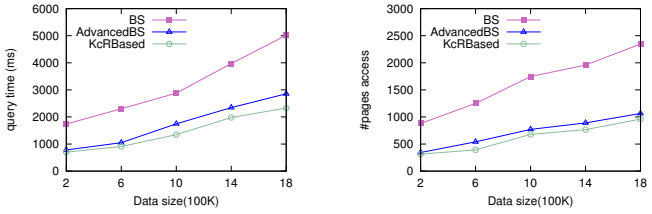
(a) Query Time (b) Penalty

Fig. 12. Varying Sample Size T

9) Approximate Algorithm: In this set of experiments, we evaluate the performance of the approximate algorithm (Section VI-B). The initial query is a top-10 query with 8 query keywords. The approximate algorithm is implemented by sampling different numbers, between 100 and 800, of keyword sets from all the candidate sets. The query time and the average penalty of the returned best refined query w.r.t. the sample size together with those of the exact algorithm are shown in Fig. 12. We can see that the query time of ***BS*** is linear to the sample size. This is because, for each sample, a spatial keyword query needs to be processed until retrieving the missing object. Moreover, for each sample size, the penalties of the refined queries are the same when using the different algorithms. This is because the sample space is the same and each algorithm returns the best refined query among the samples. We can also observe that the penalty generally decreases as the sample size increases. In particular, for ***KcRBased***, when the sample size is 800, the approximate algorithm sacrifices only 12% of the penalty while saving 30% of the query time.

10) Scalability: Finally, we test the scalability of the algorithms. We randomly select different numbers of objects from the GN dataset to evaluate the query performance under different dataset sizes. The initial spatial keyword query is a top-10 query. Fig. 13 plots the result. As we can see, the query

time and page access of the algorithms grow almost linearly when the dataset cardinality increases. Since we do not change the candidate keyword sets with the increase of the dataset size, the number of spatial keyword queries varies only little. The cost of processing a spatial keyword query increases linearly with the dataset size, which then explains the performance trend of our algorithms under different dataset sizes.



(a) Query Time (b) Page Access

Fig. 13. Varying dataset size

VIII. Conclusion

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

In future work, it is of interest to investigate the refinement of query location in spatial keyword top-k queries. Based on this, we plan to build an integrated framework that supports the answering of why-not questions on spatial keyword top-k queries while considering different parameters, including the refinement of parameter α, the query keyword set, and the location in a concerted fashion.

Acknowledgements

This work is supported by HK-RGC grants 12201615, 12202414, and 12200114. Xin Lin is the corresponding author and is supported by Shanghai Pujiang Program.

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, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu. Spatial Keyword Querying. In *ER*, pp. 16-29, 2012.

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

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

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

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

[8] L. Chen, X. Lin, H. Hu, C. S. Jensen, and J. Xu. Answering why-not questions on spatial keyword top-k queries. In *ICDE*, pp. 279-290, 2015.

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

[10] 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.

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

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

[13] Y. Gao, Q. Liu, G. Chen, B. Zheng, and L. Zhou. Answering why-not questions on reverse top-k queries. In *PVLDB*, pp. 738-749, 2015.

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

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

[16] M. Herschel and M. A. Hernández. Explaining missing answers to SPJUA queries. In *PVLDB*, 3(1-2):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. In *PVLDB*, 1(1):736-747, 2008.

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

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

[20] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-tree: an efficient index for geographic document search. In *TKDE*, 23(4):585-599, 2011.

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

[22] X. Lin, J. Xu, and H. Hu. Reverse keyword search for spatio-textual top-k queries in location-based services. In *TKDE*, to appear, 2015.

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

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

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

[26] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, pp. 205-222, 2011.

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

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

[29] D. Wu, G. Cong, and C. S. Jensen. A framework for efficient spatial web object retrieval. In *VLDBJ*, 21(6):797-822, 2012.

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

[31] 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.

[32] 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.

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

[34] 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.