YASK: A Why-Not Question Answering Engine for Spatial Keyword Query Services

Lei Chen[†] Jianliang Xu[†] Christian S. Jensen[§] Yafei Li[†]
[†]*Department of Computer Science, Hong Kong Baptist University, Hong Kong, China*
{lchen, xujl, yafeili}@comp.hkbu.edu.hk
[§]*Department of Computer Science, Aalborg University, Denmark*
csj@cs.aau.dk

ABSTRACT

With the proliferation of the mobile use of the web, spatial keyword query (SKQ) services are gaining in importance. However, state-of-the-art SKQ systems do not provide systematic functionality that allows users to ask why some known object is unexpectedly missing from a query result and do not provide an explanation for such missing objects. In this demonstration, we present a system called YASK, a whY-not question Answering engine for Spatial Keyword query services, that is capable of answering why-not questions posed in response to answers to spatial keyword top-k queries. Two explanation and query refinement models, namely *preference adjustment* and *keyword adaption*, are implemented in YASK. The system provides users not only with the reasons why desired objects are missing from query results, but provides also relevant refined queries that revive the expected but missing objects. This demonstration gives attendees hands-on experience with YASK through a map-based GUI interface in which attendees can issue spatial keyword queries, pose why-not questions, and visualize the results.

1. INTRODUCTION

The widespread diffusion of smartphones gives prominence to spatial keyword query services [2]. 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 [4].

However, due to improper system parameters or the query being issued not capturing the user's intent, a user may find that some desirable objects are unexpectedly missing from a query result. This may also make the user wonder whether other useful objects, which are as yet unknown to the user, may be missing from the result. Thus, the user has reason to question the overall utility of the query and its result. Debugging and fixing a query consumes time and may require insight that a user does not have. It is thus relevant for the system to be able to give explanations about desired but missing objects, as well as be able to automatically provide a refined query that includes the desired objects in its result.

Proceedings of the VLDB Endowment, Vol. 9, No. 13

EXAMPLE 1. *Bob visits New York for the first time, and he wants to find a nearby cafe for a cup of coffee. He issues a top-3 spatial query with keyword "coffee." However, surprisingly, the Starbucks cafe down the street, is not in the result. Bob thus wonders why the Starbucks cafe is not in the result. Are there better options? Is something wrong with the query so that other good options are also missing? How can the ranking function be adjusted so that the Starbucks cafe, and perhaps other relevant cafes, appears in the result?*

EXAMPLE 2. *In preparation for attending an overseas conference, Carol issues a query to find the top-3 hotels that are close to the conference venue and are described as "clean" and "comfortable." She is surprised that the result contains only local hotels that are unknown to her and that a well-known international hotel is not in the result. Carol wonders why this exclusion happens. Are the returned hotels really the best? Are the query keywords not properly set? How can the query keywords be minimally modified so that the expected hotel, and perhaps other good hotels, appears in the result?*

To enable the *why-not questions* [1, 3, 7] in the above scenarios, we show in prior studies that there are two possible reasons for missing objects [5, 6]. First, a missing object may be ranked very low because of an improper setting on the preference between spatial distance and textual similarity in the scoring function. For instance, the reason why Bob could not see the Starbucks cafe could be that a very low importance was given to spatial proximity in the scoring function. Second, the set of query keywords given by the user may not match the missing object well. For instance, the well-known hotel Carol could not see might be described better by "luxury"; as such, the textual relevance of this hotel to the query keywords is very low. To further follow up on such explanations, two query refinement models, namely *preference adjustment* [5] and *keyword adaption* [6], were proposed to minimally modify users' initial queries so that their expected but missing objects were returned.

In this demonstration, we present a system called YASK, a whY-not question Answering engine for Spatial Keyword query services. To the best of our knowledge, this is the first system that integrates why-not functionality into spatial keyword top-k query processing. The system provides as a web-based service. Users can issue their spatial keyword top-k queries, can ask follow-up why-not questions, and can view the results on a client browser. The query processor on the server consists of two main engines: a spatial keyword top-k query engine and a why-not question answering engine. In addition to giving explanations for the desirable but missing objects, YASK provides users with relevant, refined queries based on the two existing refinement models [5, 6], thus supporting why-not questions for spatial keyword queries.

The rest of the demonstration proposal is organized as follows. In Section 2, we give the formal definitions of spatial keyword top-k queries and why-not questions. Section 3 presents the YASK system. The interface of YASK and demonstration details are covered in Section 4.

2. QUERY DEFINITIONS

2.1 Spatial Keyword Top-k Queries

Let $\mathcal{D}$ denote a database of spatial objects. Each object $o \in \mathcal{D}$ is defined as a pair $(o.loc, o.doc)$, where $o.loc$ is the location of the object and $o.doc$ is a set of keywords that describe the object. A spatial keyword top-k query q retrieves the top-k objects from $\mathcal{D}$ ranked according to a scoring function that takes into consideration of both spatial distance and textual similarity. For broad applicability, we adopt a widely used ranking function [4]:

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

where $SDist(o,q)$ and $TSim(o,q)$ are normalized spatial distance and textual similarity, respectively. As such, a spatial keyword top-k query q takes 4 parameters $(q.loc, q.doc, k, \vec{w})$, where $q.loc$ is a query point location, $q.doc$ is a set of query keywords, k denotes the number of objects to retrieve, and $\vec{w} = \langle w_s, w_t \rangle$, where $0 < w_s, w_t < 1$ and $w_s + w_t = 1$, denotes the user preference between spatial distance and textual similarity. The distance $SDist(o,q)$ is calculated as the Euclidean distance. The textual similarity $TSim(o,q)$ can be computed using an information retrieval model. Without loss of generality, we adopt the Jaccard similarity model:[1]

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

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 $\forall o \in \mathcal{R}\ (\forall o' \in \mathcal{D} - \mathcal{R}\ (ST(o,q) \geq ST(o',q)))$.*

2.2 Why-Not Questions

It can be difficult for users to specify queries that best capture their intent. After a user issues an initial query and gets the result, the user may find that one or more objects that they expected to be in the result are missing. The user may then question the accuracy of the system and may wonder why the exclusion happens. To address such why-not questions, we adopt the query refinement approach [8] and consider two refinement models: preference adjustment and keyword adaption.

The former aims to help users adjust the preference between spatial distance and textual similarity, i.e., adjust $\vec{w}$. The latter aims to provide users with better query keywords, i.e., a $q.doc$ that better describes their intent. As simply modifying $\vec{w}$ or $q.doc$ may not bring the missing objects into the result, we also support the enlargement of k in these two models. We aim to provide the users with refined queries that minimally modify their initial queries. Specifically, each refined query q' is associated with a penalty that quantifies how different it is from the initial query.

The penalty function for adjusting the preference is defined as follows:

$$Penalty(q,q')_{\vec{w}} = \lambda \cdot \frac{\Delta k}{R(M,q) - q.k} + (1-\lambda) \cdot \frac{\Delta \vec{w}}{\sqrt{1 + q.w_s^2 + q.w_t^2}}, \quad (3)$$

where λ is a user preference of the modification on k and $\vec{w}$, $M = \{o_1, o_2, ..., o_j\}$ is the set of missing objects, and $R(M,q)$ denotes the lowest rank of the missing objects under the query q. Here, Δk is measured as $max\{0, R(M,q') - q.k\}$, since if $R(M,q') > q.k$, $q'.k$ should be set to $R(M,q')$ to achieve the lowest penalty; otherwise, $q.k$ does not need to be modified. Next, $\Delta\vec{w}$ is measured as $||q.\vec{w} - q'.\vec{w}||_2$. Further, Δk and $\Delta\vec{w}$ are normalized by $R(M,q) - q.k$ and $\sqrt{1 + q.w_s^2 + q.w_t^2}$ respectively, as they can be proved to be no larger than these two values.

[1]Other textual similarity models can also be supported [5, 6].

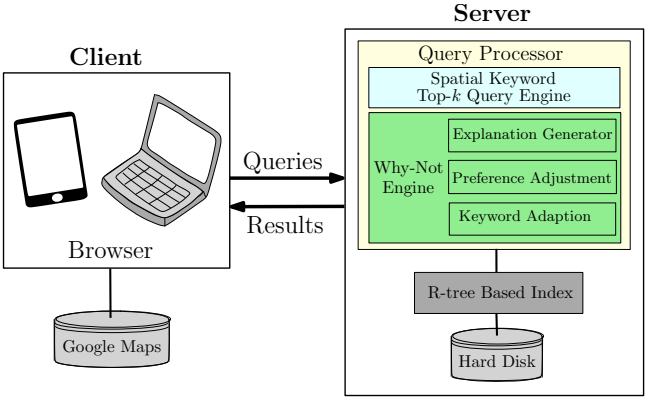

Figure 1: System Architecture of YASK

Similarly, the penalty of adapting the set of query keywords is measured as follows:

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

where Δdoc is quantified as the minimum number of operations (inserting or deleting a keyword) needed to transform $q.doc$ to $q'.doc$, which is similar to the definition of edit distance. We normalize Δdoc by the maximum possible number of edit operations needed to modify $q.doc$ to a keyword set that yields a query that retrieves all missing objects in M. This quantity is estimated as $|q.doc \cup M.doc|$, where $M.doc = \bigcup_{o \in M} o.doc$.

DEFINITION 2. ***Preference-Adjusted Why-Not Spatial Keyword Top-k Query.*** *Given a set $\mathcal{D}$ of spatial objects, a missing object set $M \subset \mathcal{D}$, and an initial spatial keyword query $q = (loc, doc, k, \vec{w})$, the preference-adjusted why-not spatial keyword top-k query returns a refined query $q' = (loc, doc, k', \vec{w}')$, with the lowest penalty according to Eqn. (3) and the result of which contains all objects in M.*

DEFINITION 3. ***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 initial spatial keyword query $q = (loc, doc, k, \vec{w})$, the keyword-adapted why-not spatial keyword top-k query returns the refined query $q' = (loc, doc', k', \vec{w})$, with the lowest penalty according to Eqn. (4) and the result of which contains all objects in M.*

3. THE YASK SYSTEM

3.1 Overview

The architecture of the YASK system is illustrated in Fig. 1. The system adopts the browser-server model. Users submit their spatial keyword top-k queries and ask follow-up why-not questions through client web browsers. The queries are sent to the server for processing. The server-side query processor consists of two engines: a spatial keyword top-k query engine and a why-not question answering engine.

The why-not engine has two query refinement modules, i.e., a preference adjustment module and a keyword adaption module, that combine to provide users with relevant refined queries. The why-not engine also has an explanation generator module that provides users with the reasons for missing but expected result objects and enables users to choose between the two refinement models. The algorithms inside the engines employ R-tree based indexing techniques [4–6]. The server processes queries using the corresponding engines and sends the results back to the users. The results are visualized in the users' web browsers using Google Maps.

3.2 Client Browser Side

The client-side browser enables users to issue queries and view their results by means of a graphical interface. It is implemented in HTML5 and JavaScript, and it uses the Google Maps API. It can be embedded into any modern web browser, such as Internet Explorer, Chrome, Safari, and Firefox.

To issue a spatial keyword top-k query, users need to input a query location, a set of query keywords, and the number of objects to retrieve. The system assumes that users have no knowledge of the ranking function used for the ranking of objects in response to a query, and it leaves the weighting vector $\vec{w}$ as a system parameter on the server. In the default setting, the spatial distance and textual similarity are weighed equally, i.e., $\vec{w} = \langle 0.5, 0.5 \rangle$. All queries are sent to the server using the standard HTTP post method.

After users get the result of an initial spatial keyword top-k query, they may find that some desired objects are unexpectedly missing from the result. In this case, users can select one or more desired objects and can then obtain information on the reason why they are missing and a refined query that includes the desired objects in its result. To explain the reason why desired objects are missing, we display analysis results on the rankings of the desired objects with regard to the initial query. To obtain a refined query, users can choose refinement of the query keywords or adjustment of the preference weighting vector. The system then returns the most relevant refined query and displays the result of the refined query. Users can apply the two refinement functions simultaneously to find better solutions.

3.3 Server Side

YASK's server side is built on Apache Tomcat, and its query engines are implemented in Java. The spatial keyword query engine processes users' spatial keyword top-k queries, and the why-not query engine implements both preference-adjusted and keyword-adapted why-not refinement models. The server caches users' initial spatial keyword queries until users give up asking follow-up "why-not" questions. In the following, we summarize the algorithms and index structures used in the engines. More technical details can be found in the literature [4–6].

Spatial Keyword Top-k Query Engine. We use an existing algorithm [4] to build the spatial keyword top-k query engine. Since the IR-tree indexing technique used in that algorithm does not support Jaccard similarity, we employ instead an indexing technique called the SetR-tree [6] with the algorithm. This technique can estimate the bound on the ranking score for all objects that are indexed by a particular tree node. Basically, each SetR-tree node has pointers to the intersection set and the union set of the keyword sets of all objects indexed by the node.

To process a spatial keyword top-k query, we maintain a priority queue $\mathcal{Q}$ that is initialized with the SetR-tree root node. In each iteration of query processing, we pop up the first element in $\mathcal{Q}$ and report it as a result if it is an object; otherwise, we unfold it and put its children into $\mathcal{Q}$. The process continues until k objects are retrieved.

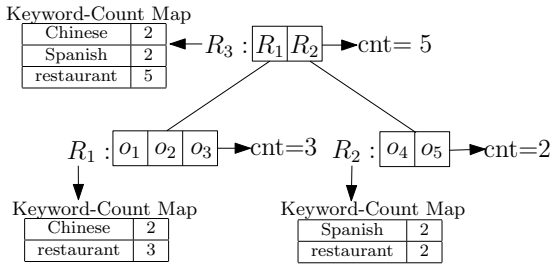

Figure 2: An Example KcR-tree

Explanation Generator Module. Given a missing object, this module generates an explanation by analyzing its spatial proximity and textual relevance with respect to the initial query based on the SetR-tree [6]. The reason can be that the missing object is too far away from the query location or that the missing object is not so relevant to the set of query keywords. The ranking of the missing object under the initial query is also provided.

Preference-Adjusted Why-Not Module. We apply a previously presented algorithm [5] to implement the preference-adjusted why-not module. The basic idea is to transform each object into a segment in a two-dimensional weight plane. As shown in [5], the best preference weighting vector must start from the origin and point to the points where the missing objects' segments intersect with other objects' segments. We use two range queries to find the segments that intersect with the missing objects' segments and compute all the intersection points. Then, with a rank update theorem [5] and the rankings of the missing objects under the initial weighting vector, we traverse all the intersection points and compute the lowest ranking of the missing objects and the penalty of the corresponding refined query. Finally, the module returns the weighting vector pointing to the intersection with the minimum penalty.

Keyword-Adapted Why-Not Module. The keyword-adapted why-not module is implemented using an optimized bound and prune algorithm [6]. The algorithm is based on an indexing structure called the KcR-tree (*Keyword count R-tree*) [6, 9]. This indexing structure is a variant of the R-tree, where each R-tree node integrates the textual information on the objects indexed in it. More specifically, each KcR-tree node is associated with a *key-value* map, where each *key* is a keyword in the union set of the keywords of the objects indexed by this node, and its corresponding value is the number of objects in this node that contain this keyword. In addition, each KcR-tree node has a cnt value that stores the number of objects that are indexed by this node. Fig. 2 shows an example of the KcR-tree. Given a KcR-tree node N, for a query keyword set $q.doc$, we can estimate the upper and lower bounds on the number of objects in N that rank higher than a missing object, and thus we can estimate the upper and lower bounds of the ranks of missing objects and the penalties of the corresponding refined query [6].

The basic idea of the keyword-adapted refinement algorithm is as follows. We generate the candidate query keyword sets and then traverse the KcR-tree starting from the root. For each candidate refined keyword set $q'.doc$, we maintain its penalty upper and lower bounds according to the ranking bounds derived from KcR-tree nodes. When traversing the KcR-tree downwards, we get tighter bounds. We prune the keyword sets whose penalty bounds exceed the currently seen best one. This process terminates when only one candidate is left, which is then returned as the result.

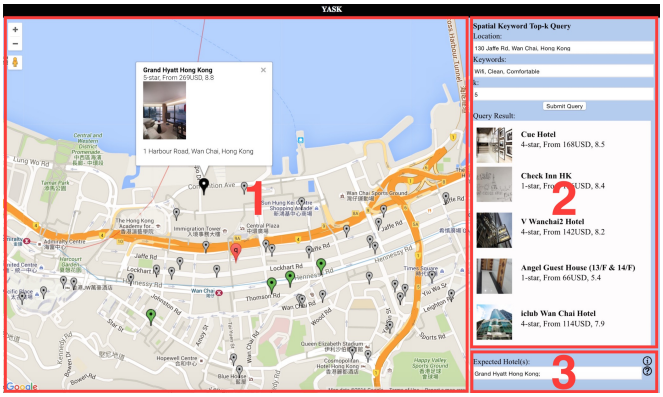

Figure 3: User Interface in the Query Mode

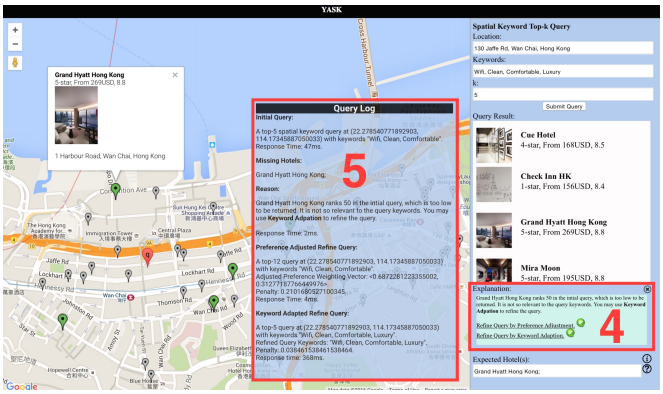

Figure 4: User Interface in the Why-Not Mode

4. DEMONSTRATION DETAILS

While the YASK system and its algorithms are built to be scalable and offer good performance for data sets with millions of objects [4–6], we use a small and focussed data set containing hotels in Hong Kong for demonstrating the system. The data set is crawled from `booking.com` and contains some 539 hotels. The keyword set for each hotel is extracted from the facilities and user comments relating to the hotel. A video of YASK can be found at `https://youtu.be/XINMX9LTSQg`. We showcase the following scenarios.

Spatial Keyword Top-k Querying. The main interface of YASK consists of three panels as shown in Fig. 3. Panel 1 displays an interactive map interface. Initially, all hotels on the map in Panel 1 are marked with grey markers. Using Panel 2, users can issue an initial spatial keyword top-k query by entering a query location, a query keyword set, and a result set size. The query location can be input by typing an address in the corresponding text box or by simply clicking on the map. We indicate the query location by a red marker on the map. The query results are indicated by green markers. Users can also browse the results in the result window in Panel 2.

Interacting with Why-Not Questions. After issuing an initial query, users can ask follow-up why-not questions through Panel 3, also shown in Fig. 3. Desired hotels can be selected by entering their names or by clicking on their markers on the map. We highlight users' expected but missing hotels with black markers. Then users can obtain explanations for why the hotels were missing by

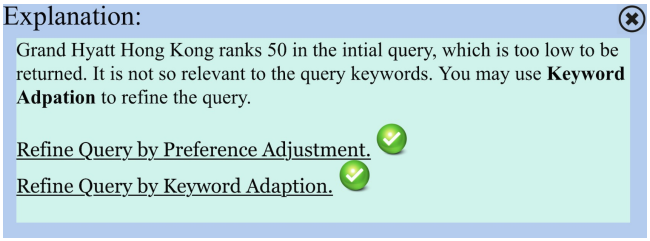

Figure 5: Explanation Panel

clicking on the ⓘ icon in Panel 3. When this is done, an explanation panel will pop up (Panel 4 in Fig. 4).

In the explanation panel (Fig. 5), users can also choose to refine the initial query by either preference adjustment or keyword adaptation. The refined query with updated input parameters is shown, and its result is displayed. As shown in Fig. 4, users can also view detailed query information (Panel 5) from the query log by clicking on the ⓘ icon. Here, users can find the detailed parameter settings for the refined query, its penalty against users' initial queries, as well as the query response time.

Query Refinement Effectiveness. We also demonstrate the effectiveness of the YASK system in answering why-not questions. In particular, we are able to show how the initial queries are *minimally* modified to revive the missing hotels and to demonstrate the impact of the setting of weight parameter λ in the penalty functions (Eqns. (3) and (4)) on the quality of refined queries.

Acknowledgements

This work was partially supported by Mr. Kwok Yat Wai and Madam Kwok Chung Bo Fun Graduate School Development Fund and HK RGC grants 12201615 and 12200114. Part of Chen Lei's work was done when he was visiting Aalborg University.

5. 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] 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.

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

[4] 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.

[5] 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.

[6] L. Chen, J. Xu, X. Lin, C. S. Jensen, and H. Hu. Answering why-not spatial keyword top-k queries via keyword adaption. In *ICDE*, 2016.

[7] 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.

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

[9] X. Lin, J. Xu, and H. Hu. Reverse keyword search for spatio-textual top-k queries in location-based services. In *TKDE*, 27(11):3056–3069, 2015.