SpaceKey: Exploring Patterns in Spatial Databases

Yixiang Fang[1], Reynold Cheng[1], Jikun Wang[1], Budiman[1], Gao Cong[2], Nikos Mamoulis[3]

[1]*The University of Hong Kong,* [2]*Nanyang Technological University,* [3]*University of Ioannina*

[1]{yxfang,ckcheng,jkwang,budiman}@cs.hku.hk, [2]gaocong@ntu.edu.sg, [3]nikos@cs.uoi.gr

***Abstract*— Spatial objects associated with keywords are prevalent in applications such as Google Maps and Twitter. Recently, the topic of spatial keyword queries has received plenty of attention. Spatial Group Keyword (SGK) search is a popular class of queries; their goal is to find a set of objects which are close to each other and are associated to a set of input keywords. In this paper, we propose *SpaceKey* [1], a system for retrieving and visualizing spatial objects returned by SGK queries. In addition to existing SGK query types, *SpaceKey* supports a novel query, called *SPM query*. An SPM query is defined by a *spatial pattern*, a graph whose vertices contain keywords and its edges are associated with distance constraints. The results are sets of objects that match the pattern. *SpaceKey* allows users to perform comparison analysis between different SGK query types. We plan to make *SpaceKey* an open-source web-based platform, and design API functions for software developers to plug other SGK query algorithms into our system.**

I. Introduction

With the prevalence of location-based services in many real applications [1], [2], [3] such as Google Maps, Flickr, and Twitter, spatial keyword queries have received plenty of research interest in recent years. Spatial Group Keyword (SGK) search [4], [5] is an important class of spatial keyword queries. Existing query types in this class (e.g., mCK [4]) aim at finding a set of objects, which are relevant to a set of input keywords, and whose locations are close to each other. However, the distance constraints that user may want to specify could be more general. For example, when a user wishes to rent/buy a house, she may expect a school nearby, which is close to her house but not too close (e.g., to avoid the noise and crowd caused by school), so their distance should be less than $1km$, but larger than $0.3km$. Moreover, although there are several existing SGK query types and systems [6], [7], [8] that support them, there is a lack of a platform for visualizing and analyzing the query results. Thus, it is hard to compare them systematically.

To tackle issues above, we first present the *spatial pattern matching* (SPM) query, a new type of SGK queries. SPM queries are based on the concept of *spatial pattern*, a graph whose vertices contain keywords and their edges are labeled with distance interval constraints, capturing the distance constraints among the objects that instantiate the vertices of the patterns. The goal is to find all the matches (i.e., sets of objects) that satisfy a given spatial pattern. A detailed definition of SPM queries and evaluation algorithms can be found in [9]. The main subject of this paper is *SpaceKey*, an online system that allows users to express various SGK queries (including SPM query), and visualize and analyze their results, in a simple and interactive manner.

[1]We have submitted the video about *SpaceKey*, and the video can also be accessed from http://i.cs.hku.hk/~yxfang/demo.mp4

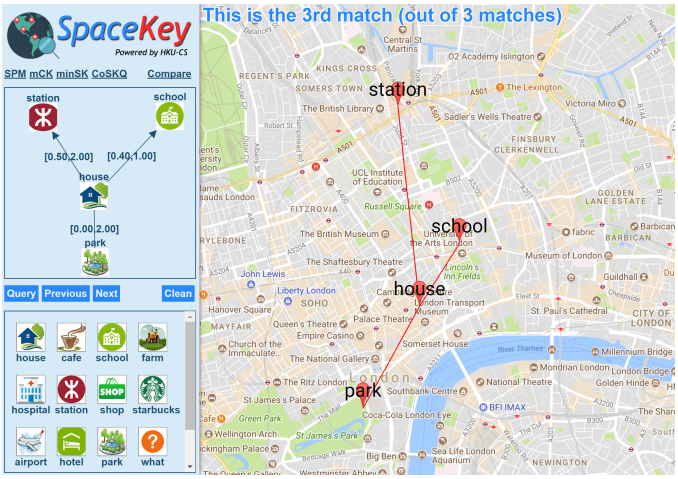


Fig. 1. User interface of *SpaceKey*.

In *SpaceKey*, to issue an SPM query, the user can easily draw a spatial pattern and view the query results. Figure 1 shows the user interface of *SpaceKey* configured to run on a dataset of UK Points of Interest (PoIs). The left panel shows a spatial pattern of a house, a school, a station, and a park with some distance intervals. Once the user clicks the "Query" button, all the matches will be returned and the user can view them one by one through the "Previous" and "Next" buttons. Additionally, *SpaceKey* allows a query user to edit a previous spatial pattern, which would help her to interactively compose a new query pattern and explore the matched objects.

Besides, *SpaceKey* can seamlessly supports other SGK algorithms. Currently, we have incorporated three additional SGK query types: mCK [4], CoSKQ [5], [10], and minSK [11]. Thus, a user can choose which query model to use to fit her needs. In Figure 1, a query user can issue a specific query after clicking its type on top of the left panel. Moreover, *SpaceKey* provides an Application Programmer Interface (API), which consists of a list of functions. To plug a new SGK query type or algorithm into *SpaceKey*, the user only needs to follow the API and slightly modifies the HTML codes in the panel under the logo, and then she can easily view the query results and compare them with other SGK query algorithms. Furthermore, *SpaceKey* is an open-source software, so application developers can customize *SpaceKey* to suit their own needs.

Furthermore, our system provides a user-friendly interface

that enables online analysis of results returned by different SGK queries. It can report statistics about the results returned by different algorithms, such as the number of object sets returned, the average pair-wise distance between objects in each set, and the diameter of objects. These features allow users to perform a detailed comparison between the results output from different algorithms. Additionally, users can plug in their own analysis functions through the API provided.

In summary, our system incorporates a novel query, called SPM query. Moreover, it enables a clear visualization of the differences among different SGK query algorithm, which could work as a tool for users to pick the right solution. In addition, *SpaceKey* provides a list of API functions, which facilitates installation and testing of new SGK solutions. Our system will be valuable to users who are interested in SGK solutions (e.g. database and GIS researchers, application developers), and work as a demo for corporations that are interested to implement related functions.

The rest of the paper is organized as follows. In Section II, we review related work and position *SpaceKey* in it. In Section III, we introduce the SPM query. Section IV describes the framework of *SpaceKey*. In Section V, we explain how we are going to demonstrate *SpaceKey*.

II. Related Work and Novelty

A. SGK Queries

There are two types of SGK queries in the literature. The first type of queries takes as arguments a set of keywords and returns a group of objects [4], [11] that are close to each other, and which are related to the set of query keywords. A representative query is the m-closest keyword (mCK) query [4], which finds a group of objects that collectively contain all the m query keywords, and the maximum distance between any two objects returned is minimized. Its variants include minSK [11] which minimizes a different distance cost function. The second type (e.g., [5], [10]) takes as input the location where the query is issued, and a set of keywords. A list of objects is returned, each of which is near to the query location and is relevant to the keywords. A presentative query type is CoSKQ [5], [10]. A more detailed discussion of these SGK queries can be found from [9].

B. Systems for Spatial Keyword Queries

Cao et al. [6] present SWORS, a spatial web object retrieval system for retrieving objects satisfying spatial keyword queries. In [7], a web-based service, called GroupFinder, which is able to return top-k groups of PoIs according to a scoring function, is proposed. Chen et al. [8] developed SOPS, an application for providing spatial-keyword publish/subscribe service over a stream of geotextual objects. In [12], a system called RISE for region search and exploration on spatial objects is proposed. Chen et al. [13] developed the YASK system for answering why-not questions posed in response to answers to spatial keyword top-k queries.

Although these systems perform well in particular scenarios, there is a lack of tools for integrating various SGK algorithms as well as performing comparison analysis. Moreover, these systems cannot support SPM queries and it is not clear how to plug new SGK queries into these systems. To tackle these issues, we develop the *SpaceKey* system.

III. The SPM Query

We consider a database D of spatial objects (or *objects* for brevity). Each object $o \in D$ has 2D coordinates (o_x, o_y), and is associated with a set, $doc(o)$, of keywords. We say that the object o *matches* with a keyword w, if $w \in doc(o)$. Now we formally introduce the spatial pattern and SPM query.

Definition 1 (spatial pattern)**.** *A spatial pattern P is a simple graph (V, E) of n vertices $\{v_1, v_2, \cdots, v_n\}$ and m edges, and the following constraints hold:*

- *Each vertex $v_i \in V$ has a set of keywords w_i;*
- *Each edge $(v_i, v_j) \in E$ has a distance interval $[l_{i,j}, u_{i,j}]$, where $l_{i,j}$ ($u_{i,j}$) is the lower (respectively upper) bound of distances between two matching objects in D;*
- *Each edge $(v_i, v_j) \in E$ is associated with one of the signs: (1) $v_i \rightarrow v_j$; (2) $v_i \leftarrow v_j$; (3) $v_i \leftrightarrow v_j$; and (4) v_i–v_j.*

The meanings of these signs on the edges are as follows.

• $v_i \rightarrow v_j$ [v_i excludes v_j]: No object with keyword w_j in D should have a distance less than $l_{i,j}$ from o_k.

• $v_i \leftarrow v_j$ [v_j excludes v_i]: No object with keyword w_i in D should have a distance less than $l_{i,j}$ from o_l.

• $v_i \leftrightarrow v_j$ [mutual exclusion]: No object with keyword w_i in D should have a distance less than $l_{i,j}$ from o_k, and the distance of any object with keyword w_j in D should be at least $l_{i,j}$ away from o_l.

• v_i–v_j [mutual inclusion]: The occurrence of any object (other than o_k and o_l) with keywords w_i and w_j in D with distance shorter than $l_{i,j}$ is allowed.

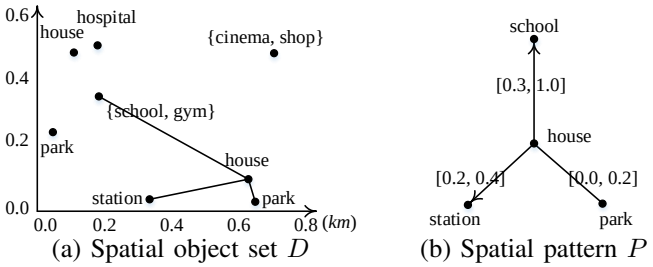
Suppose a user wishes to retrieve two objects (say, o_s and o_t) such that: (1) o_s and o_t have keywords *house* and *school* respectively; (2) the distance of o_s from o_t is between $0.3km$ and $1.0km$; and (3) there does not exist any object with keyword *school*, which is less than $0.3km$ from o_s. Then, these requirements can be expressed as an edge *house*$\rightarrow$*school* with distance interval [0.3, 1.0] (*km*), as shown in Figure 2(b).

We say that, two objects o_k and o_l constitute an *e-match* of (v_i, v_j), if o_k and o_l match with w_i and w_j respectively, and they satisfy the distance constraints of (v_i, v_j). Based on the concept of e-match, we define the *match* as follows.

Definition 2 (match)**.** *Given a spatial pattern P and a set S of objects, S is a match of P if (1) for each edge of P, there is an e-match in S; and (2) there does not exist any proper subset S' of S such that for each edge of P, there is an e-match in S'.*

Problem 1 (Spatial Pattern Matching query)**.** *Given a spatial pattern P, SPM returns all the matches of P in D.*

In Figure 2, given a spatial pattern in Figure 2(b), the SPM query returns one match, which contains four objects connected in solid lines, as shown in Figure 2(a).



(a) Spatial object set D (b) Spatial pattern P

Fig. 2. Illustrating the SPM query.

To our best knowledge, no previous work offers a workable solution to SPM queries. In [9], we show that the SPM problem is NP-hard. To answer the SPM queries efficiently, we propose two efficient algorithms. The first one follows the steps of classical multi-way join, which generally is easy to understand, but not efficient enough. The second one, called `MSJ`, finds all the e-matches in a collective manner, and our experimental results [9] on real large datasets show that it performs best; therefore in *SpaceKey* we use `MSJ`.

IV. System Overview

We illustrate the system framework of *SpaceKey* in Figure 3. It adopts the browser-server model. The *Browser* side provides interfaces for users to submit queries and view the query results and statistics of them for comparison. To issue a query, the user can simply draw a spatial pattern or input some query keywords via the browser. Then, the query is sent to the server for processing, after which the results will be returned and displayed on the browser.

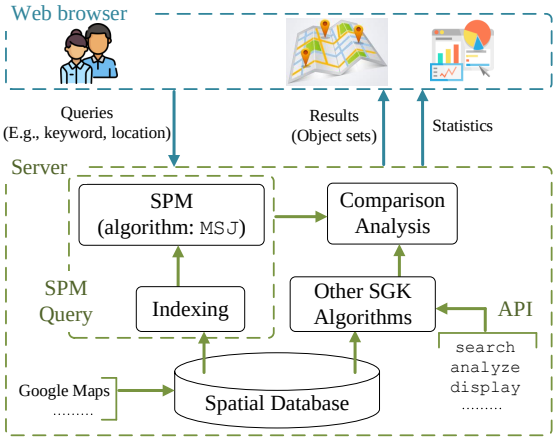


Fig. 3. The framework of *SpaceKey*.

On the *Server* side, there are two modules, i.e., *SPM Queries* and *Comparison Analysis*, used for answering SPM queries online and analyzing comparison respectively. Since the SPM query algorithms [9] rely on the IR-tree index [14], we build the IR-tree for the objects offline in the *Indexing* module.

The *Comparison Analysis* module performs comparison analysis for the results returned by different SGK query algorithms, and reports their statistics (e.g., the number of objects and the average distance between each pair of objects). Moreover, users can easily plug their own SGK query algorithms into the system, and view and compare the statistics

```
public interface Algorithm {
      public List<Object> search(Query query);
      public void analyze(List<Object> objs);
      public void display(List<Object> objs);
      …
}
```

Fig. 4. The API functions of *SpaceKey*.

with the help of our easy-to-use Java API functions. Some typical API functions are discussed in Section IV-A. Currently, in *SpaceKey* we have implemented three other SGK query algorithms, which are mCK [4], CoSKQ [5], [10], and minSK [11]. We remark that *SpaceKey* employs modular design, which may facilitate the addition of future extensions.

A. API

Our system provides a Java Interface, which consists of a list of API functions. For public users, to plug in their own query algorithms, they just need to implement the functions of the interfaces with their own algorithms, and slightly modify the web page codes. Then they can visualize and compare the results. Figure 4 shows three typical API functions:

- `search`: it performs query processing of an SGK query.
- `analyze`: it analyzes the results of an SGK query.
- `display`: it generates the HTML codes for visualizing the query results of an SGK query in the webpage.

V. Demonstration

A. Setup

We implemented the algorithms in Java and designed the system using JavaServer Pages (JSP) technique with the Tomcat server. We used a dataset, obtained from *www.pocketgpsworld.com*, which contains PoIs (e.g., cinemas) in UK. Each object in the dataset has several types (e.g., "park") which are used as keywords of the object, and a pair of latitude and longitude values denoting its location. The dataset contains 182,317 objects in total and 45,317 distinct keywords.

B. Demonstration

1) SPM Queries: To issue an SPM query, a user first needs to draw a spatial pattern. Specifically, the user can create some vertices by dragging the icons from the bottom left panel as depicted in Figure 1. Note that if the user wants to specify keywords that do not appear in the bottom left panel, she can simply edit icons by changing their keyword sets. Then, to link two vertices, the user just needs to choose one vertex and click the other one. Also, the user can specify the distance constraint of an edge by double clicking it as shown in Figure 5. After that, by clicking the "Query" button, the user submits an SPM query. The server will process the query and show the first match, marked with red balloons, in the map of the right panel.

2) Other SGK Query Algorithms: In *SpaceKey*, we have implemented three other SGK query algorithms, i.e., mCK, minSK, and CoSKQ. To issue a SGK query, a user first needs to click its name at the top of the left panel, and then the interface of query parameters will appear in the left panel. For an mCK or a minSK query, a user only needs to specify some keywords, while the CoSKQ query takes a location with

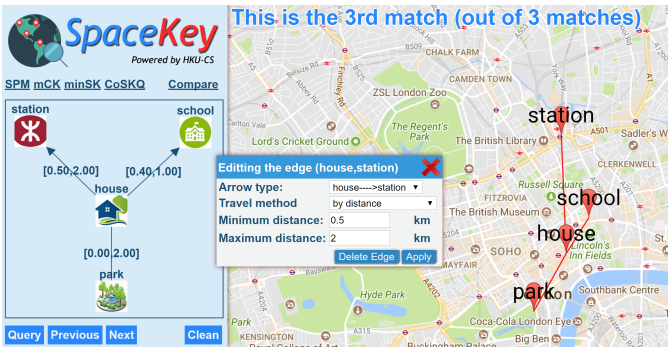


Fig. 5. Editing a spatial pattern.

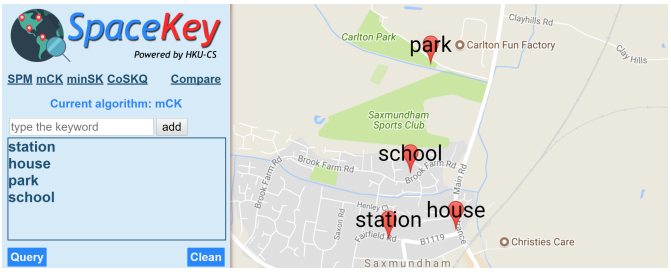


Fig. 6. The user interface of the mCK query.

a set of keywords as its input. Figure 6 shows the user interface of the mCK query. We skip the interface of minSK since it is almost the same with that of mCK. For the CoSKQ query, its input contains a set of keywords, and a query location. To input the location, a user can click any location on the map and its latitude and longitude will be recorded in the input boxes automatically. After clicking the "Query" button, the query results will be displayed on the map. Figure 7 depicts the query interface, where the query location is also marked.

Fig. 7. The user interface of the CoSKQ query.

3) Comparison Analysis: Our system allows users to compare the query results returned by different SGK queries in terms of various statistics, including the number of matched object sets, average pairwise distance, diameter and so on. By clicking the "Compare" button, a user can step into the pop-up window to perform comparison analysis. As shown in Figures 8 and 9, a user first needs to select the algorithms to execute and compare, then specifies the query keywords (and locations), and finally clicks "Compare". After that, a statistics table will appear, showing various statistics mentioned above. In addition, the user can click the names of the SGK queries in the bottom to view the results in the map instantly.

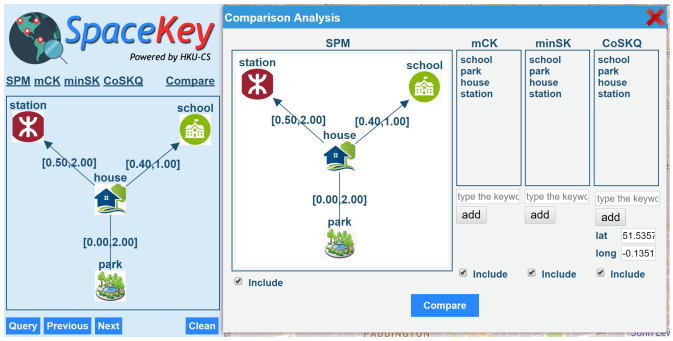


Fig. 8. The user interface for comparison analysis.

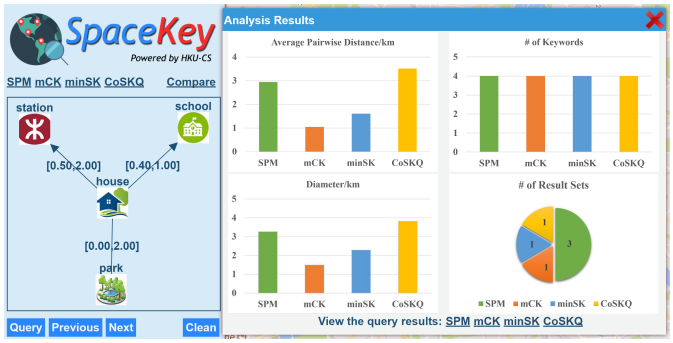


Fig. 9. The results of comparison analysis.

ACKNOWLEDGMENTS

Reynold Cheng, Yixiang Fang, Jikun Wang, and Budiman were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 104004572, 102009508 ,104004129). Nikos Mamoulis has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 657347.

REFERENCES

[1] Y. Fang et al, "Scalable algorithms for nearest-neighbor joins on big trajectory data," *TKDE*, vol. 28, no. 3, pp. 785–800, 2016.
[2] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *PVLDB*, vol. 9, no. 12, pp. 1233–1244, 2016.
[3] Y. Fang et al, "Effective community search over large spatial graphs," *PVLDB*, vol. 10, no. 6, pp. 709–720, 2017.
[4] T. Guo, X. Cao, and G. Cong, "Efficient algorithms for answering the m-closest keywords query," in *SIGMOD*. ACM, 2015, pp. 405–418.
[5] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *SIGMOD*. ACM, 2011, pp. 373–384.
[6] X. Cao et al., "Swors: A system for the efficient retrieval of relevant spatial web objects," in *PVLDB*, vol. 5, no. 12, 2012, pp. 1914–1917.
[7] K. S. Bøgh, A. Skovsgaard, and C. S. Jensen, "Groupfinder: A new approach to top-k point-of-interest group retrieval," *PVLDB*, 2013.
[8] L. Chen, Y. Cui, C. Gao, and X. Cao, "Sops: A system for efficient processing of spatial-keyword publish/subscribe." in *PVLDB*, 2014.
[9] Y. Fang et al, "On spatial pattern matching," http://i.cs.hku.hk/~yxfang/spm.pdf, Under review in ICDE 2018.
[10] C. Long, R. C. W. Wong, K. Wang, and A. W. C. Fu, "Collective spatial keyword queries: A distance owner-driven approach." in *SIGMOD*, 2013.
[11] D. Choi, J. Pei, and X. Lin, "Finding the minimum spatial keyword cover," in *ICDE*. IEEE, 2016, pp. 685–696.
[12] K. Feng, K. Zhao, Y. Liu, and G. Cong, "A system for region search and exploration." in *PVLDB*, vol. 9, no. 13, 2016, pp. 1549–1552.
[13] L. Chen, J. Xu, J. Christian, and Y. Li, "Yask: A why-not question answering engine for spatial keyword query services." in *PVLDB*, 2016.
[14] D. Wu et al., "Joint top-k spatial keyword query processing," *TKDE*, vol. 24, no. 10, pp. 1889–1903, 2012.