A Map Search System based on a Spatial Query Language

Yuanyuan Wang
Yamaguchi University, Japan
y.wang@yamaguchi-u.ac.jp

Panote Siriaraya
Kyoto Sangyo University, Japan
spanote@gmail.com

Haruka Sakata
Kyoto Sangyo University, Japan
g1544647@cc.kyoto-su.ac.jp

Yukiko Kawai
Kyoto Sangyo University/Osaka University, Japan
kawai@cc.kyoto-su.ac.jp

Keishi Tajima
Kyoto University, Japan
tajima@i.kyoto-u.ac.jp

ABSTRACT

We propose a novel query language that can express complex spatial queries in a concise and intuitive way for map search. The proposed language can express conditions on the range, direction, and time distance within their spatial search queries. In this language, we introduce several spatial operators, such as "space character" operator which is used to represent the geographical distance between matching objects in a concise and intuitive way as well as arithmetic and directional operators which enables the combination and manipulation of spatial areas. We also show how a map search system supporting this query language can be implemented, and describe several applications created using this system to highlight how the query language can be put into practice. These applications include a web interface which allows developers to embed a function of spatial search by our query language into their systems. In addition, we developed a mobile Android application that allows nonprofessional users to easily search for nearby venues and routes by using the proposed query language. Finally, we outline the results of a study carried out to evaluate the potential usefulness of our proposed search system.

1 INTRODUCTION

In map search systems, keyword-based queries are widely adopted due to their ease of use. Such queries are however inadequate when more complex requests are needed. For example, consider the case where a software tool would need to be developed to help identify appropriate locations for real estate development. Such a software needs to deal with search requests which contain multiple constraints and spatial conditions. For instance, those interested in constructing a family-friendly apartment building would search for vacant locations that have many schools and playgrounds within traveling distance, but are far enough away from inappropriate locations or noisy public spaces.

In map search systems with simple keyword-based queries, such a search task requires multiple query transactions by user operations. For example, to find a good apartment for a family, the transactions need to (1) search for apartments by using a keyword query, (2) also identify schools on the map, (3) limit the query result to those within 4km from the found schools, and (4) also exclude those which are within 200m from some inappropriate locations. The same is true for various complex requests such as "Finding all the restaurants located between two famous tourist landmarks" (when developing an application for tourism) or "Finding all the shops located next to parks in a city" (for location-based recommender systems). As shown in these examples, it is difficult to process complex search requests using simple keyword-based queries. Systems only supporting such keyword-based queries require multiple steps including non-textual interactions to process them.

Figure 1: Examples of map search using spatial operators.

On the other hand, there have been much research on spatial logic or algebra. In a spatial database of PostGIS[1], the spatial search task with location queries can be run in SQL, and they can represent complex spatial conditions in queries, but they require users to learn and understand the progrmming-language-like syntax, and as a result, they are too complicated for general users in many applications. They are, therefore, impractical for the use in such systems.

The goal of this research is to design a simple spatial query language which can represent such complex queries within a single query statement with a concise, and more intuitive syntax so that users can easily specify complex queries. Our language uses 10 spatial operators to represent conditions on directions, ranges, angles, and time distance, which allow users to incorporate spatial conditions and manipulation of spatial areas within their keyword-based-like queries. Fig. 1 shows examples of map search carried out by queries including spatial operators in our language, such as ([^] [-] [*]). For example, Fig. 1 (c) shows the result of a query ("my house"␣0.5km␣"pizza parlor") * ("friends house"␣0.5km␣"pizza parlor").

2 RELATED WORK

As explained in Section 1, there have been much research on spatial databases based on region algebra or region logic. However, most current commercial location-based services such as Google Maps or Bing Maps are designed mainly to help general users to execute two types of simple tasks: (1) find certain places and locations within a specified geographical area and (2) find the best route (e.g., shortest distance, most economical) between two given locations. Recent academic research in this domain also mainly focus on similar problems, such as locating comfortable, aesthetically pleasing or safe routes [2] and personalizing the search results by identifying locations which better match the latent interests of users [5].

[1] https://postgis.net/

Table 1: Spatial operators by using the space-key

Operator	␣*	␣^	␣_	␣@	␣[x-y]	␣$	␣#
Processing	Surrounding	Direction (north/up)	Direction (south/down)	Angle	Range	Size	Time

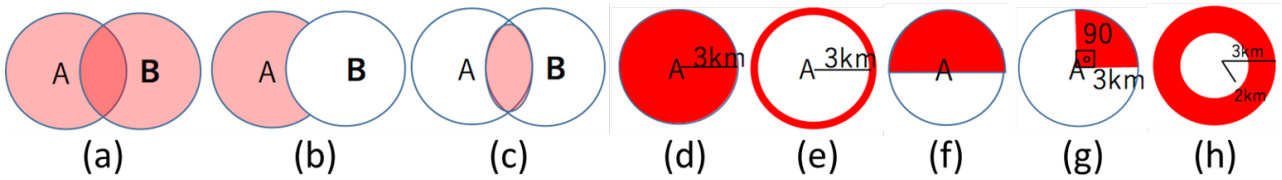

Figure 2: (a) Union (b) Difference (c) Intersection (d) Within (e) Distance (f) Direction (north) (g) Angle (90 degree) (h) Range

Because most of the current commercial systems are focusing on general users executing these types of tasks, they use simple keyword-based queries instead of complicated query languages. For example, if users need to search for restaurants in London, they would simply input a query such as "Restaurants in London". For users who need more complex and precise search requests, however, such search systems are inadequate. Although various API systems[2] [3] have been created to provide access to the more advanced features of the location search systems, they are often limited to single-process tasks (finding locations within a specific distance, geo-coding a specific location name, etc.). Therefore, in this paper, we propose a novel query language to enhance the existing API systems, which allows users to express conditions on distance, space and time distance towards objects matching the query keywords through the use of spatial operators.

While the use of operators in keyword-based queries for map search is not common, the use of operators in keyword-based queries can be found in other domains. In document search, proximity operators have been proposed as a way to limit search results to those that contain the keywords in a specific order or within a specific distance [3]. Also in video database domain, Pradhan et al. [4] proposed operators which allow users to represent constraints on pairs of matching objects to be joined. In graph database domain, Cypher[4] is a declarative query language that allows users to state what actions they want performed upon their graph data without requiring them to describe (or program) exactly how to do it.

3 THE SPATIAL QUERY LANGUAGE

A description of our proposed spatial query language is described in this section. Overall, there are two main rules which are used to define the syntax for a spatial search unit in our query language.

Rule 1: The syntax of the most primitive unit of spatial queries is defined as follows: "A␣spatial length␣α" A and α are keywords, with A denoting the location of the origin for the spatial search for the object with the property α. It is permissible to either denote the spatial length by using a unit distance (i.e., 200m, 3 minutes) or by using continuous spaces. When continuous space is used, the spatial length would represent the N nearest locations with the property α, where N is represented by the number of continuous spaces.

(Example) : A␣800m␣α denotes a query statement to identify the α objects which exists inside the region 800m from the origin point A.

(Example) : A␣␣␣␣α denotes a query statement to identify the nearest 3 α objects from the origin point A.

Rule 2: The keywords (e.g., A and α) used in the primitive unit of the spatial query would be encapsulated within a double quotation mark (e.g., "Tokyo tower" or "Grand Central Terminal, New York" for A or "pizza shop" or for α).

In addition, various spatial, directional and distance operators can be used to impose conditions when conducting a spatial search.

Rule 3: Each primitive spatial search Unit can be combined with other units through the use of spatial, directional and distance operators in a mathematical equation format.

(Example) : (A␣800m␣α) + (B ␣300m␣α)

3.1 Spatial Operators

The standard set operators can also be used as spatial operations for the query unit defined previously. These include the union [+], difference [-], and intersection [*]. Users can use such operations to manipulate the spatial region they wish to search into. Examples of queries including these operators are as follows:

Union calculation (Ex.1) : (A␣3km␣α)+(B␣3km␣α) denotes the union of the spatial region *within* 3km from point A AND the spatial region *within* 3km from point B (see Fig. 2 (a)).

Difference calculation (Ex.2) :(A␣3km␣α)-(B␣3km␣α) denotes the spatial region *within* 3km from point A which is NOT *within* 3km from point B (see Fig. 2 (b)).

Intersection calculation (Ex.3) :(A␣3km␣α)*(B␣3km␣α) denotes the spatial region that is *within* 3km from point A which is ALSO *within* 3km from point B (see Fig. 2 (c)).

All set operators (+, -, *) can be used to search for objects with the properties identified in the query unit. For example, the aforementioned (A␣3km␣α)*(B␣3km␣α) query would search for objects with the property α which is located within the spatial region that is the result of the intersection between 3km from points A and B.

3.2 Directional and Distance Operators

Table 1 shows the 7 spatial operators which can be used to further denote distance and direction within our spatial query. Examples of four expressions which use these operators are described below:

Distance operation (Ex.4) : A␣3km␣α

[2] https://www.microsoft.com/en-us/maps/choose-your-bing-maps-api
[3] https://cloud.google.com/maps-platform/
[4] https://neo4j.com/cypher-graph-query-language/

retrieves the objects α which are *within* 3km from point A (Fig. 2(d)).

Within operation (Ex.5) : A␣*3km␣α
retrieves the objects α that are 3km *away from* point A (Fig. 2 (e)).

Direction operation (Ex.6) : A␣^3km␣α
retrieves the objects α which are to *the north* of, and within 3km from, point A (see Fig. 2 (f)).

Angle operation (Ex.7) : A␣3km@90␣α
retrieves the objects α which exist *within* 3km in the 90 *degree* counterclockwise direction from point A (see Fig. 2 (g)).

3.3 Range, Size, and Time Operators

Our proposed spatial query language also includes a variety of range operators which allows users to more accurately specify the desired search range within their query (see Table 1). For example:

Range operation (Ex.8) : A␣[1km-3km]␣α
retrieves the objects α in the region from 1km to 3km from point A (see Fig. 2 (h)).

In addition, size [$] and time operators [#] can be used to formulate search queries. The size operator $ extracts the size of the corresponding property of object α and uses it as a unit of measure (i.e., 1 city block = 0.5km). The same is true for time operations depend on the context (i.e., using A␣#3min to find venues which are 3 minutes walking distance from point A, when a user selects a "walking" direction).

4 SPATIAL SEARCH SYSTEM

In this section, we explain the structure of our search system. The system consists of three main components: (1) an Input/Output component that processes user requests and outputs them to the appropriate format; (2) an interpreter component that parses and processes queries and (3) the data processing program component to link the search system to appropriate data sources.

Users of our system would send an HTTP request to the server with details of the spatial query as parameters. The query specified by the client is then passed to the interpreter and the data processing components. These components would parse the query, process the request and send the results back to the Web In/Output Processing component which would transmit the results back to the client as an HTTP response in a data format such as JSON or XML.

The role of the interpreter component is to process the spatial operators sent as the request. This component consists of a query parser, a spatial data converter, and a spatial data calculator. For the parser, the role is to analyze the user query and determine the appropriate operations and procedures to process it. For example, the following query: (A␣^3km␣α) + (B␣^3km␣α) would be processed by the parser into the following steps: Var1= A␣^3km␣α (step 1), Var2= B␣^3km␣α (step 2), and Result=Var1+Var2 (step 3).

These steps would then be processed by the interpreter. Each spatial variable is sent to the data converter to convert the elements (such as A␣^3km or B␣^3km) in the query to spatial regions which represents the correct distribution of those elements. The conversion program would access information provided by the data processing component to calculate the appropriate regions. For example, when processing the element "A␣^3km␣shops", the data processing component would calculate the geographical location of point A as well as the geographical

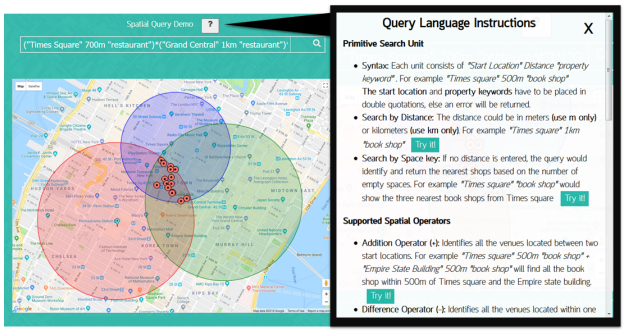

Figure 3: Spatial query language demo application [5].

locations of shops within a 3km radius. After the data has been converted, the spatial operators are then processed. For example, if the request query contains the intersection operator [*], it would calculate the region which is the overlap between the converted A␣3km and B␣3km regions. After all the calculations have been completed, the result is sent back to the client in the appropriate data type (JSON or XML etc.) as specified by the data processing component.

A prototype of the search system engine was implemented as a RESTFUL web service using nodeJS. The current system supports spatial map search, with the input being the requested as a spatial query (an HTTPS GET request) and the output is an array of locations which match the spatial query (returned using the JSON data format) together with details such as the name of the place and the address. The equation expression within the spatial query was parsed using the Shunting-yard algorithm. Google Maps API was used in the data processing proportion to identify the various locations specified in the primitive spatial query unit (i.e., "times square") and their geographical positions. The system could also later be easily adapted to utilize other data sources such as Open Street Map data or a customized SQL database as well.

5 DEMONSTRATION APPLICATIONS

To highlight how the system could be useful in practice, a number of web applications were created which utilized our proposed spatial search query language and would be shown during the demonstration. The first application was a web interface for our search system engine which users could use to test the query language or search for locations using spatial equations[3]. Users would be able to use the spatial, range and directional operations described in Section 3 as well as mathematical expressions such as brackets to compose their search queries. After clicking the search button, the system would send the user's query to the search system server and would then render the search results received from the server onto the map. For example, Fig. 3 shows the results of the query: (("Times Square"␣700m␣"restaurant")*("Grand Central"␣1km␣"restaurant"))*("Pennsylvania station"␣1km␣"restaurant"), which aims to identify all restaurants located within 700m of Times Square and 1km from Grand Central and Pennsylvania station. One potential use-case for such a query is for example to identify potential meeting places for three users based on their starting locations (For example, when one user works near Times Square and the other near Grand Central and the final near Pennsylvania station and the system

[5] http://yklab.kyoto-su.ac.jp/~sakata/spatialQueryDemo

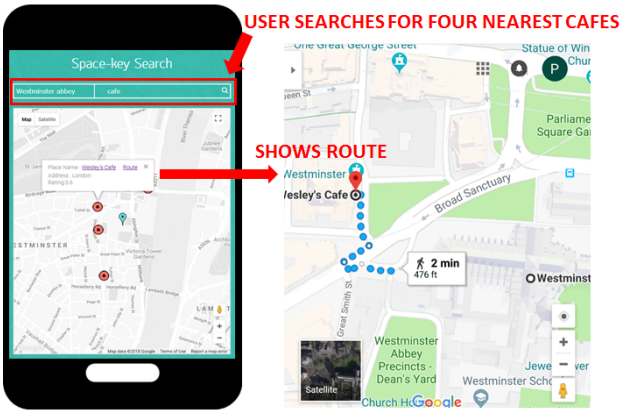

Figure 4: Space-key search application for novice users.

would need to find a restaurant that is equally near to all three of their workplaces for them to meet for lunch). The web interface system also provides an instruction page where the various operators in our query language are explained and a number of examples shown (see Fig. 3). Users could click on the "try it" button to examine the search results of the examples and could also freely modify the example equations.

Furthermore, another application which utilized our proposed query language (only using the "space-key" for a more simple and intuitive search) would also be shown during the demonstration. This application was conceptualized by looking at how non-professional common users generally used location-based mapping services. Although route navigation was a commonly used feature, users also generally used location-based services to quickly identify different types of nearby venues and then find out how they could travel to such locations. Therefore, we developed an android application ("space-key search" application) which utilized the primitive unit of our spatial query language to allow users to search for nearby venues (users are able to search for nearby locations using only the space-key). For example, the user would enter the query "Current Location␣␣␣␣␣Restaurants", to find the four nearest restaurants to them on auto adjust map zooming (see Fig. 3). Clicking on the markers would show details of the venue (the address, review scores etc.) as well as a link with the details of the route to the store. The application itself could be downloaded from Google play store[6]. A mobile web version of this application [7] was also developed for evaluation and demonstration purposes (Fig. 4 shows screen-shots of the applications). A demonstration video of the applications discussed in the paper which would be presented at the conference could be viewed from the following link [8].

6 USER STUDY

A user evaluation study was also carried out to evaluate the potential usefulness of our proposed query language. The main aim was to determine whether such an equation based query language would be feasible for developers to learn and use. 15 students from a university-level computer science course were recruited and asked to carry out a series of location search tasks. Each participant was asked to use both our proposed query language through the web interface system which was developed (spatial language condition) as well as through the Google Maps system (Google map system) to complete 5 search assignments. Each search assignment consisted of a task to search for places (e.g., pizza parlors) near a specific location (e.g., The White House). For example, one task consisted of trying to find the number of pizza parlors located within 500m of Times Square. In another task, participants were asked to find the number of pizza parlors located within 700m of Times Square which is also located 1000m from Empire State Building. Written instructions and examples were provided to help participants complete the tasks and introduce the various spatial operators. An objective measurement of performance was obtained by measuring the time users spent on each task. To measure subjective user experience, the System Usability Scale (SUS) was used [1], which involved the rating of perceived effectiveness, efficiency, and satisfaction.

Overall, participants rated a higher SUS score for the spatial language condition (Mean=70.17, SD=13.09) than the Google map condition (Mean=24.46, SD=10.61) ($t(13)=7.24$, $p<0.001$)). In addition, participants were able to complete the tasks using less time (seconds) in the spatial language condition (Mean=82.27, SD=28.18) then the Googlemaps condition(Mean=180.86,SD=49.25) ($t(13)=-8.219$, $p<0.001$). Therefore, it seems that at least for search tasks which involve the combination and manipulation of spatial regions, the proposed map search system could indeed be useful.

7 CONCLUSION

In this paper, we proposed a novel query language for spatial search which can be used to express complex queries in a text equation format. We implemented a prototype of our proposed system and developed two applications (a web based application and a mobile application) to showcase how the query language could be put into practice. In addition, we conducted a user study, The results of which highlights the potential usefulness of our query language.

In the future, we look to expand our query language to other search domains such as text and video search. Although we have shown how our language could be used in map search, our proposed query language could easily be applied to spatial search within documents and videos as well. For example, a query searching for text within a document that contains the word "B" and is within 5 sentences from the word "A" is expressed by "A␣5sentences␣B".

ACKNOWLEDGMENT

The work in this paper is partially supported by JSPS KAKENHI Grant Numbers 16H01722, 15K00162, 17K12686.

REFERENCES

[1] John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.
[2] Jaewoo Kim, Meeyoung Cha, and Thomas Sandholm. 2014. SocRoutes: Safe Routes Based on Tweet Sentiments. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14 Companion)*. ACM, New York, NY, USA, 179–182. https://doi.org/10.1145/2567948.2577023
[3] Elsevier Newsletter. 2015. How Can I Search Literature with Reduced Noise? Utilization of "Proximity Operator" in ScienceDirect & Scopus. (August 19 2015).
[4] Sujeet Pradhan, Keishi Tajima, and Katsumi Tanaka. 2001. A query model to synthesize answer intervals from indexed video units. *IEEE Transactions on knowledge and data engineering* 13, 5 (2001), 824–838.
[5] Hongzhi Yin, Yizhou Sun, Bin Cui, Zhiting Hu, and Ling Chen. 2013. LCARS: A Location-content-aware Recommender System. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, New York, NY, USA, 221–229. https://doi.org/10.1145/2487575.2487608

[6] https://play.google.com/store/apps/details?id=com.kawaiLab.spatialQuery
[7] https://yklab.kyoto-su.ac.jp/~sakata/simple/spatialQuary/
[8] http://yklab.kyoto-su.ac.jp/SpatialDEMO/Spatial_demo_movie.mp4