Designing a Query Language Using Keyword Pairs for Spatial and Temporal Search

Yuanyuan Wang[1](✉), Panote Siriaraya[2], Haruka Sakata[2], Yukiko Kawai[2,3], and Keishi Tajima[4]

[1] Yamaguchi University, 2-16-1 Tokiwadai, Ube, Yamaguchi 755-8611, Japan
y.wang@yamaguchi-u.ac.jp
[2] Kyoto Sangyo University, Motoyama, Kamigamo, Kita-ku, Kyoto 603-8555, Japan
spanote@gmail.com, kawai@cc.kyoto-su.ac.jp
[3] Osaka University, 5-1 Mihogaoka, Ibaraki, Osaka 567-0047, Japan
[4] Kyoto University, Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan
tajima@i.kyoto-u.ac.jp

Abstract. Our goal is to design a query language based on two keywords for spatial and temporal search by using a single textual query in an intuitive way. This language as a form of syntactic sugar can express complex spatio-temporal queries which include conditions on range, direction, time length and size which are difficult to express through textual queries in conventional keyword-based search systems. To express these conditions, our proposed language introduces 12 spatio-temporal operators such as arithmetic and directional operators which enables users to combine and manipulate spatial and temporal areas. Also, we use "space characters" (the space-key) between keywords which are used to express the geographical distance or time-length between matching objects in an intuitive way for general users. In this paper, we provide an overview of our proposed search system where we can retrieve maps and web documents to highlight how the query language could be put into practice by using complex spatio-temporal queries. Finally, we discuss the results of a user study carried out to evaluate the potential usefulness of our proposed search system.

Keywords: Query language · Spatial and temporal search · Space-key search · Spatio-temporal operators

1 Introduction

Although nonverbal search services continue to increase in modern society, have you ever been to conscious about using a space-key (␣) while typing search queries? Probably "no" because space characters do not usually have important meaning in keyword search other than in the conjunction of the components. We propose to give a valuable and useful role to space characters in search queries

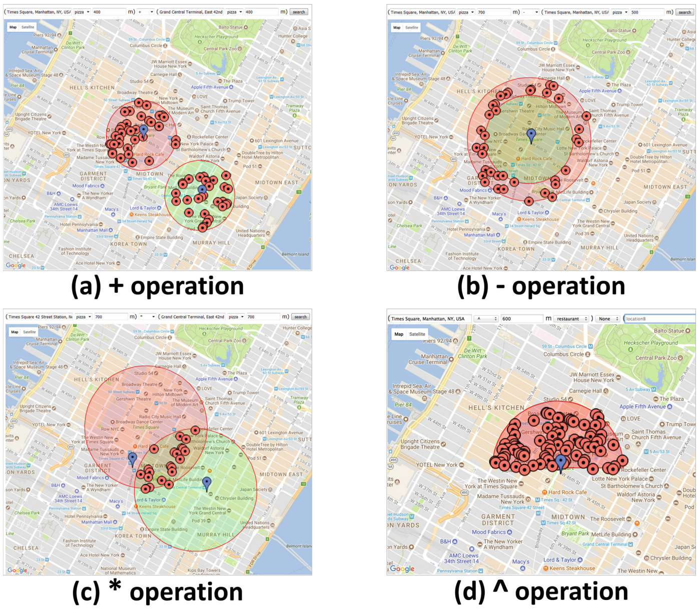


Fig. 1. Example of map search using proposed operators.

based on two keywords. Space characters are useful and powerful for expressing spatio-temporal relationship in queries for maps, images, videos, and web pages, while it is easy to use and intuitive for users.

While, simple keyword-based queries have been widely adopted in search systems because of their ease of use, they are not good at expressing spatial and temporal relationships. Such queries are difficult to find appropriate maps, images, and videos, when more complex requests are needed. For example, when we travelling by car, to find a famous pizza parlor within 100–500 m from our current position on a map service supporting keyword queries, we need four transactions: (1) search for pizza parlors on the map by using a keyword query, (2) also identify current position on the map, (3) limit the query result to those which are within 500 m from the current position, and (4) also exclude those which are within 100 m from the current position. Queries including more complex conditions are even harder to execute in a single transaction. Such queries include: ($Q1$) "Find all pizza parlors which are within 500 m from our current position and within 500 m from location B", ($Q2$) "Find all pizza parlors which are within the range of 100–500 m from my current position", ($Q3$) "Find all pizza parlors that are within 500 m from my current position which are also within 500 m from location B", and ($Q4$) "Find all pizza parlors located to the north of and are also within 500 m from my current position". 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 or need to use very complicated query languages to process them.

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, they require users to learn and understand the programming-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.

In this paper, we propose a query language based on two keywords that expresses spatial queries within a single query statement with a concise and intuitive syntax so that general users can easily specify complex queries. Our approach uses the length of spaces between two keywords in queries to express conditions on geographical distances and time-length between objects matching the two keywords. This allows us to express queries as a form of syntactic sugar including complicated spatio-temporal conditions in a very simple and intuitive way. In the conventional systems, queries "A␣B" and "A␣␣␣␣␣␣B" return the same results containing both "A" and "B". In our query language however, these two queries have a different meaning: the latter means "Search the 6 nearest B objects from A". For example, a query searching for B within 5 km from A on the map is expressed by "A␣5km␣B". A similar spatio-temporal operation can also be used for retrieving documents and video. For example, a query searching for a document containing "B" within 5 sentences from "A" is expressed by "A␣5 sentences␣B".

Our language also introduces 12 operators including set operators and spatio-temporal operators to express queries including conditions on directions, ranges, angles and time. Figure 1 shows an example of map search using operators ([+] [−] [*] [ˆ]). For example, $Q1$ shown above can be expressed by using a union operator ([+]) as follows: (A␣800m) + (B␣800m)␣pizza parlor, which aims to identify all the pizza parlors found within the combined region which is within 800 m from A and within 800 m from B (see Fig. 1(a)). $Q2$ can be expressed by using a range operator ([−]) as follows: (A␣800m) − (B␣600m)␣pizza parlor, which aims to identify all the pizza parlors located within 800 m from A and excluding those which are located within 600 m from B (see Fig. 1(b)). $Q3$ can be expressed by using a set operator ([*]) as follows: (A␣0.5km␣pizza parlor) * (B␣0.5km␣pizza parlor), which aims to identify all pizza parlors located within 0.5 km both from A and from B (see Fig. 1(c)) and $Q4$ can also be expressed by using directional condition operators ([ˆ] or [θ]) as follows: A␣ˆ500m␣pizza parlor, which aims to identify all pizza parlors located to the north of and are within 500 m from A (see Fig. 1(d)). We also introduce the property operator ([$]), which is used to test properties of objects matching to query keywords. For example, a query "current position␣500m$20people␣pizza parlor" retrieves pizza parlors within 500 m which can cater to 20 people.

The remainder of this paper is structured as follows. In Sect. 2, we discuss previous research which has been carried out related to map, text and video search.

[1] https://postgis.net/.

Afterward, in Sect. 3, we provide a definition of spatio-temporal operators and provide examples of how they could be applied to query data. Section 4 describes the structure and components of a spatio-temporal search system. Section 5 provides application examples of an implemented spatio-temporal search system. Section 6 discusses our proposed search method with a user study. Finally, in Sect. 7, we conclude this paper and discuss future works.

2 Related Work

2.1 Map Search

Most of the current major location services such as Google Maps or Bing Maps focus on finding certain locations within a specified geographical area or the best routes (e.g., shortest distance, most economical) matching given query keywords. For example, when you want to find restaurants in Kyoto station, a query "restaurant␣in␣Kyoto station" could be used to find restaurants in Kyoto station, and when you want to find supermarkets around Tokyo station, the query "supermarket␣near␣Tokyo station" could be used to find supermarkets around Tokyo station. In regards to finding appropriate routes, there have been many studies aimed at helping users locate simple and memorable [3,19], comfortable [10,15,17], safe [5,9,12], or aesthetically pleasing routes [16] in map services, as well as studies which seek to personalize the search results, by identifying locations which better match the latent interests of users [11,21].

These search systems are often designed for general users and they only utilize keyword matching algorithms to process user search queries. For example, users can simply input a query such as "supermarket in Tokyo station" to search for supermarkets around Tokyo station. However, these systems cannot satisfy users when they wish to retrieve more complex and precise search requests. Although several API systems, i.e., Bing Maps APIs[2] and Google Maps Platform[3] provide access to more advanced features of map search systems, they are often limited to single-process tasks (e.g., finding places within a specific distance, geocoding a certain location name). Therefore, in this paper, we propose a novel spatio-temporal query language as a form of syntactic sugar that can express conditions on distance and time-length between objects matching the query keywords in an intuitive way for general users, and we also show the demonstration for map search. For example, we can issue a query "Tokyo station␣500m␣supermarket" to find supermarkets within 500 m from Tokyo station.

2.2 Spatio-Temporal Search for Text and Video

Proximity operators are used in some document search systems or Web search engines to narrow search results by limiting them to those that have certain query keywords placed within a specified distance or in the specified order. For example,

[2] https://www.microsoft.com/en-us/maps/choose-your-bing-maps-api.
[3] https://cloud.google.com/maps-platform/.

ScienceDirect and Scopus utilize a proximity operator (W/n) which retrieve documents where query keywords appear within n words [4]. For example, a query "microscopy W/3 gfp" could be used to find literature containing the word "microscopy" that is within 3-word lengths of the word "gfp". Oracle Text utilize a proximity operator (near) which requires the query keywords to appear within a default number of word distances [8]. For instance, a query "near((dog, cat), 6)" could be used to search for all documents which include the word "dog" that is within 6-word lengths of the word "cat". In our case, we express constraints on the distance between query keywords (or objects matching query keywords) by using space characters (␣) between the query keywords, allowing general users to easily express queries that include spatio-temporal conditions.

Video interval operators such as union, intersection, concatenation, and their set-variants have been devised to allow users to programmatically edit videos [7,20]. Although researchers have defined several interval operations to compute new intervals from existing intervals, the new intervals cannot satisfy users' search requests as they lack certain types of algebraic operators such as difference. However, given a set of fragmentarily indexed video shots, these operations generally produce fragmentary intervals and thus cannot always produce appropriate intervals which users generally intend to find. Pradhan et al. also proposed interval operators called *glue join* operations for composing longer intervals from a set of short annotated intervals [13,14,18]. These studies focus on using operators to compose semantically meaningful video intervals and determines how to efficiently evaluate them. On the other hand, the purpose of our research is to design a query syntax that can express spatio-temporal conditions in an easy and intuitive way.

3 Spatio-Temporal Query Language

In this research, we define 12 operations including set operations and spatio-temporal operations by text for use in search operations. We mainly show examples taken from map search tasks, but we also show examples from Web search and video search tasks.

Definition 1: The syntax of the most primitive unit of the spatio-temporal query is defined as follows: A␣spatio-temporal length␣α. A and α are keywords, with A denoting the location of the origin for a spatio-temporal search for the object with the property α. This permits us to use more than two contiguous spaces from the first appearing space (␣) by using the space-key. The part corresponding to "spatio-temporal length (spatio-temporal operator)" should follow after the last space (␣) in those contiguous spaces. When continuous spaces are used, the spatio-temporal 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 exist inside the region 800 m from the origin point A.

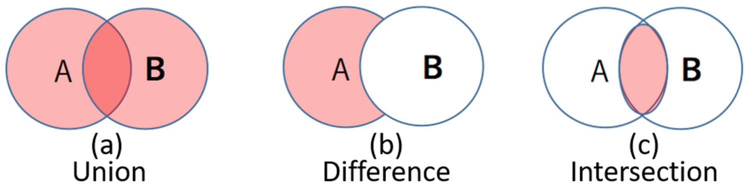


Fig. 2. Set operators.

Table 1. Spatial operators by using the space-key

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

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

Definition 2: The spatio-temporal length used in the syntax above are represented by a unit distance (e.g., 100 m, 1 sentence, etc.). For example, the unit distance "800 m" could be used following the ␣ spatio-temporal operator.

Definition 3: The keywords (e.g., A and α) used in the primitive unit of the spatio-temporal 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 could be used to impose conditions when conducting a spatio-temporal search.

Definition 4: Each primitive spatio-temporal search unit could be combined with other units through the use of spatial, directional, and distance operators in a mathematical expression format as follows: $SQ^1 \cup SQ^2 \cup ... = SQ^+$.
(Example): (A␣800m␣α) + (B␣500m␣α)
denotes a query to identify the union of the α objects within 800 m from point A and the region within 500 m from point B.

3.1 Set Operators

The standard set operators can also be used as spatial-temporal operations for the query unit defined previously [1,6]. These include the union [+], difference [−], and intersection [*] operators. Users could use such operations to manipulate the spatial region they wish to search into. Examples of queries including these operators are shown below:

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

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

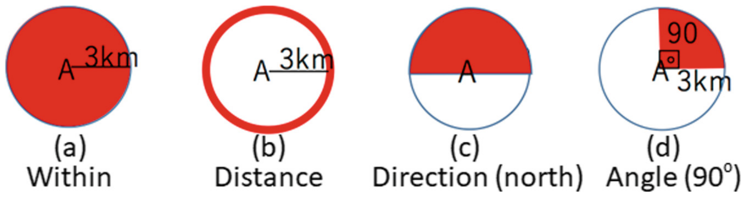


Fig. 3. Spatial operators.

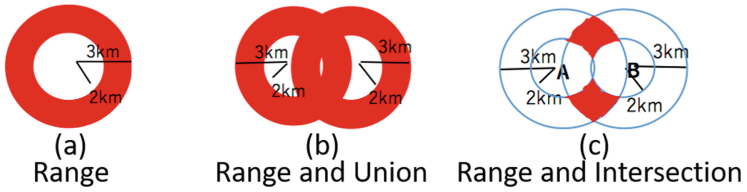


Fig. 4. Range operators.

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

All set operators (union [+], difference [−], intersection [*]) 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 3 km from points A and B. Note that only union [+] and intersection [*] can be used between two search objects if the search objects have different types (e.g., restaurant and hotel).

3.2 Spatial Operators

Table 1 shows the 7 spatial operators which could be used to further denote distance and direction within our proposed spatial-temporal search queries. Examples of four expressions which use these operators (within, distance, direction, and angle) are described below:

Within operation (Ex.4): A␣3km␣α
retrieves the α objects which are *within* 3 km from point A (Fig. 3(a)).
Distance operation (Ex.5): A␣*3km␣α
retrieves the α objects that are 3 km *away from* point A (Fig. 3(b)).
Direction operation (Ex.6): A␣^3km␣α
retrieves the α objects which are to the *north* of, and *within* 3 km from, point A (see Fig. 3(c)).
Angle operation (Ex.7): A␣3km@90␣α
retrieves the α objects which exist within 3 km in the 90° counterclockwise direction from point A (i.e., is east of point A) (see Fig. 3(d)).

Table 2. Time operators by using the space-key

Operator	␣#*	␣#<	␣#>	␣#[x-y]	␣#$
Processing	Around	Before	After	Range	Current

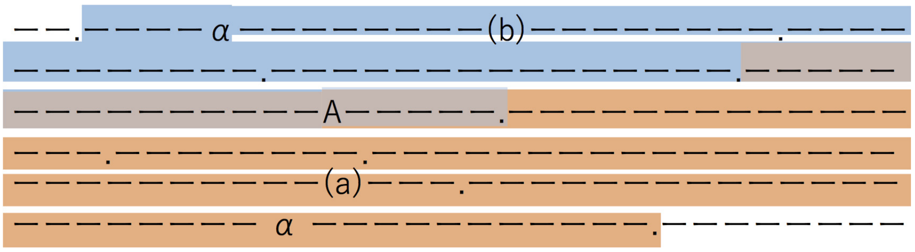


Fig. 5. Example of document extraction by Web search.

3.3 Range Operators

Next, we show three examples of the range operations shown in Table 1.

Range operation (Ex.8): A␣[3km-1km]␣α
retrieves the α objects in the spatial region from 1 km to 3 km from point A (see Fig. 4(a)). The same condition can also be expressed by the following query, which uses the "difference" operator [−]: (A␣3km␣α) − (A␣1km␣α).

Range operation (Ex.9): (A␣[3km-1km]␣α) + (B␣[3km-1km]␣α)
retrieves the α objects which exist inside the spatial region from 1 km to 3 km either from point A and from point B (see Fig. 4(b)).

Range operation (Ex.10): (A␣[3km-1km]␣α) * (B␣[3km-1km]␣α)
retrieves the α objects which exist inside the spatial region from 1 km to 3 km both from point A and from point B (see Fig. 4(c)).

3.4 Size Operators

Below is an example query including the size operator shown in Table 1. The size operator [$] extracts the size of the corresponding property of object α and uses it as a unit of measure (e.g., 1 city block = 0.5 km). This operator is placed immediately after the corresponding object size.

Size operation (Ex.11): A␣3km$10units␣$\alpha$
retrieves the α objects that have a size of 10 units *within* 3 km from point A.

3.5 Time Operators

Time operator [#] shown in Table 1 can be used to formulate time-based search queries. Table 2 shows five different time operators which could be used. For the map search, the time operators express conditions on time distance from point A to α (e.g., the time needed to travel from point A to the α objects). The computation of the route and the required time for the route is system dependent (see Sect. 4 for more details).

Time operation (Ex.12): A␣#3min␣α
retrieves the α objects that are *within* 3 min from point A.
Time operation (Ex.13): A␣#*3min␣α
retrieves the α objects that are 3 min *away from* point A.
Time operation (Ex.14): A␣^#3min␣α
retrieves the α objects that are to the north of, and *within* 3 min of point A.
Time operation (Ex.15): A␣#3min@90␣α
retrieves the α objects that are *within* 3 min from point A and ALSO in the rotated 90° counterclockwise from the east direction of point A.

3.6 Applied to Web and Video Search

In addition to the map search, our proposed spatio-temporal query language could also be applied to Web search and video search.

Application Examples for Web Search. The spatio-temporal query language expresses conditions to extract documents from point A to the α objects.

Within operation: A␣3lines␣α
retrieves the lines including the α objects which are *within* the third line from point A (see Fig. 5(a)).
Range operation: A␣*3lines␣α
retrieves the lines including the α objects that are 3 lines *away from* point A (see Fig. 5(a)(b)).
Direction operation: A␣#<3lines␣α
retrieves the lines including the α objects which are *before* 3 lines from point A (see Fig. 5(b)).

Here, the α object can also be used as an extension (e.g., .png) or a link.

Direction operation: A␣#<3lines␣.jpg
retrieves the .jpg files (images) which are *before* 3 lines from point A.
Direction operation: A␣#<3lines␣URL
retrieves the URL links which are *before* 3 lines from point A.

In addition to searching the line, a paragraph could also be searched. Furthermore, it is possible to perform a search that takes visual direction into account by using DOM analysis. This allows a visual search of where on the page the object exists.

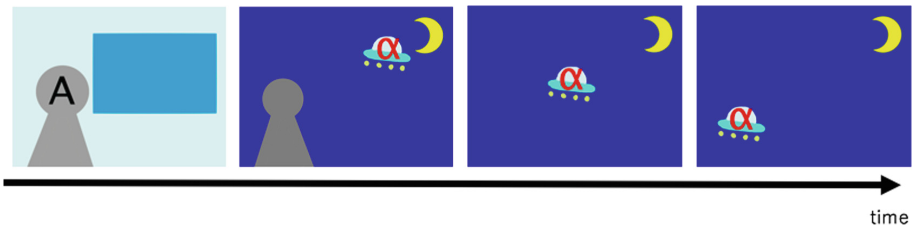


Fig. 6. Example of interval extraction by video search.

Direction operation: A␣^3lines␣.jpg
retrieves the .jpg files (images) which are *above* of and is *within* 3 lines from point A.

Range operation: A␣[@180-@90]␣.jpg
retrieves the .jpg files (images) which layout in the position from the rotated 90° counterclockwise to the rotated 180° counterclockwise from the east direction (left side) of point A.

Application Examples for Video Search. The spatio-temporal query language expresses conditions on video frame extraction from point A to the α objects.

Time operation: A␣#3min␣α
retrieves the α video frames that are *within* 3 min from point A.

Time operation: A␣#*3min␣α
retrieves the α video frames that are 3 min *away from* point A.

Time operation: A␣#<3min␣α
retrieves the α video frames which are *before* 3 min from point A.

Furthermore, it is possible to perform a search that takes visual direction into account by video frame analysis. This allows a visual search of where on the video it exists.

Direction operation: A␣^3min␣α
retrieves the α video frames which are *above* of and *within* 3 min from point A.

Direction operation: A␣[@270-@90]␣α
retrieves the video frames contain α which exist in the screen from the rotated 90° counterclockwise to the rotated 270° counterclockwise from the east direction of point A (see Fig. 6).

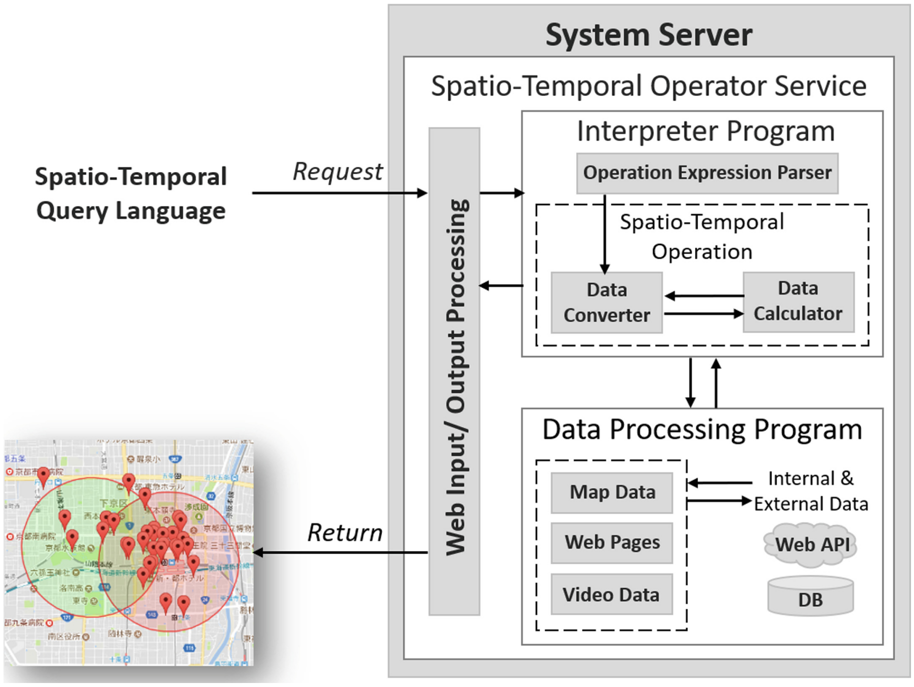


Fig. 7. Overview of the spatio-temporal search system.

4 Spatio-Temporal Search System

In this section, we explain the structure of our proposed spatio-temporal search system (see Fig. 7). The system consists of three main components: (1) a Web Input/Output component that processes user requests and output them to the appropriate format; (2) an interpreter component that parses and processes queries including the spatio-temporal operators; and (3) the data processing program component which is developed by the system developer to link the spatio-temporal search system to appropriate data sources.

4.1 Web Input/Output Processing

By using our system, a client sends an HTTP request to the server with details of the query including spatio-temporal operators and requested data type (map, web page, video) as API parameters. For example, the client retrieving map information by a spatial query A␣^3km␣α sends the following GET request to the server: www.hoge.com/map/?q=A++^3km+α (or l=A&space=3km&q=α).

The data type and 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 format for API.

4.2 Interpreter and Data Processing

The role of the interpreter component is to process the spatial-temporal operators sent as the request from the users. This component consists of a query parser, a spatio-temporal data converter, and a spatio-temporal 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, when users input the following query: (A␣^3km␣α * B␣^3km␣α) + (C␣^5km␣β + D␣^1km␣β).

It would be processed by the parser into the following steps:

Var1 = A␣^3km␣α * B␣^3km␣α (step 1)
Var2 = C␣^5km␣β + D␣^1km␣β (step 2)
Result = Var1 + Var2 (step 3)

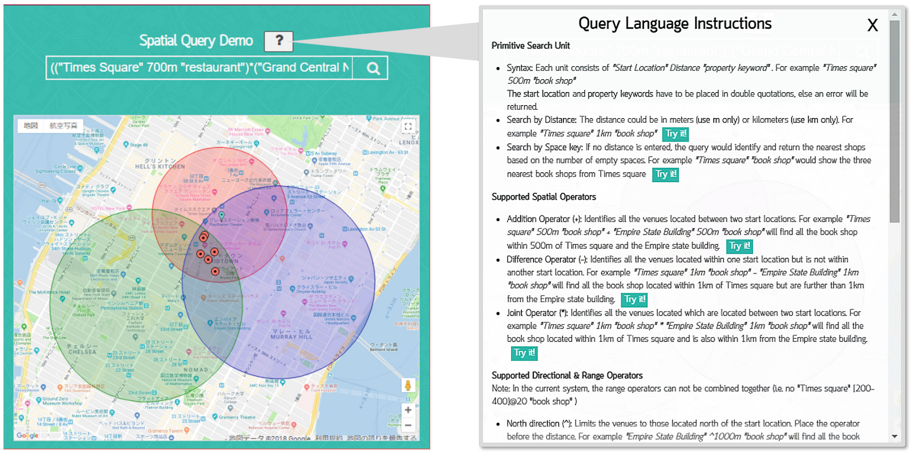
Next, these steps would then be processed by the interpreter. Each spatial variable is sent to the data converter to convert the elements (i.e., A␣^3km or B␣^3km) in the query to spatial regions which represents the correct distribution of those elements. The conversion program would access information provided by the data processing component to calculate the appropriate regions. For example, when processing the element "A␣^3km␣shops" for a map, the data processing component would calculate the geographical location of point A as well as the geographical locations of shops within a 3 km radius. If the system developer has specified Google API as the data source, the corresponding data would be obtained through a Web request sent to the API servers. Alternatively, if the developer specified an internal SQL database as the data source, the corresponding data are obtained through an SQL request. The same process would also be carried out for the other elements "B␣^3km␣α", "C␣^5km␣β", and "D␣^1km␣β".

After the data has been converted, the spatial operators are then processed. For example, if the request query contains the intersection operator [*] in step 1, it would calculate the spatial 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.

5 Demonstrative Applications

A prototype of the spatio-temporal search system[4] was implemented as a RESTFUL Web service using Node.js. The current system supports spatial map search, with the input being the requested as a spatial query (an HTTPS GET request) and the output being an array of locations which match the spatial query (returned using the JSON data format). The operation 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 (e.g., "Times Square") and their

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



Query Language Instructions

Primitive Search Unit

- Syntax: Each unit consists of *"Start Location" Distance "property keyword"*. For example *"Times square" 500m "book shop"*
  The start location and property keywords have to be placed in double quotations, else an error will be returned.
- Search by Distance: The distance could be in meters (use m only) or kilometers (use km only). For example *"Times square" 1km "book shop"* Try it!
- Search by Space key: If no distance is entered, the query would identify and return the nearest shops based on the number of empty spaces. For example *"Times square" "book shop"* would show the three nearest book shops from Times square Try it!

Supported Spatial Operators

- Addition Operator (+): Identifies all the venues located between two start locations. For example *"Times square" 500m "book shop" + "Empire State Building" 500m "book shop"* will find all the book shop within 500m of Times square and the Empire state building. Try it!
- Difference Operator (-): Identifies all the venues located within one start location but is not within another start location. For example *"Times square" 1km "book shop" - "Empire State Building" 1km "book shop"* will find all the book shop located within 1km of Times square but are further than 1km from the Empire state building. Try it!
- Joint Operator (*): Identifies all the venues located which are located between two start locations. For example *"Times square" 1km "book shop" * "Empire State Building" 1km "book shop"* will find all the book shop located within 1km of Times square and is also within 1km from the Empire state building. Try it!

Supported Directional & Range Operators

Note: In the current system, the range operators can not be combined together (i.e. no "Times square" [200-400]@20 "book shop")

- North direction (^): Limits the venues to those located north of the start location. Place the operator before the distance. For example *"Empire State Building" ^1000m "book shop"* will find all the book

Fig. 8. Spatial query language demo application[4].

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.

5.1 Applications of Map Search

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. The first application was a Web interface for our search system which users could use to test the query language or search for locations using the spatial query language. Users would be able to use the spatial, range, and directional operations described in Sect. 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 return the search results received from the server onto the map. For example, Fig. 8 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 700 m of Times Square and 1 km from Grand Central and Pennsylvania station. One potential use-case for such a query is to create a “meet up feature”, which would 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 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. 8). Users could click on the “Try it!” button to examine

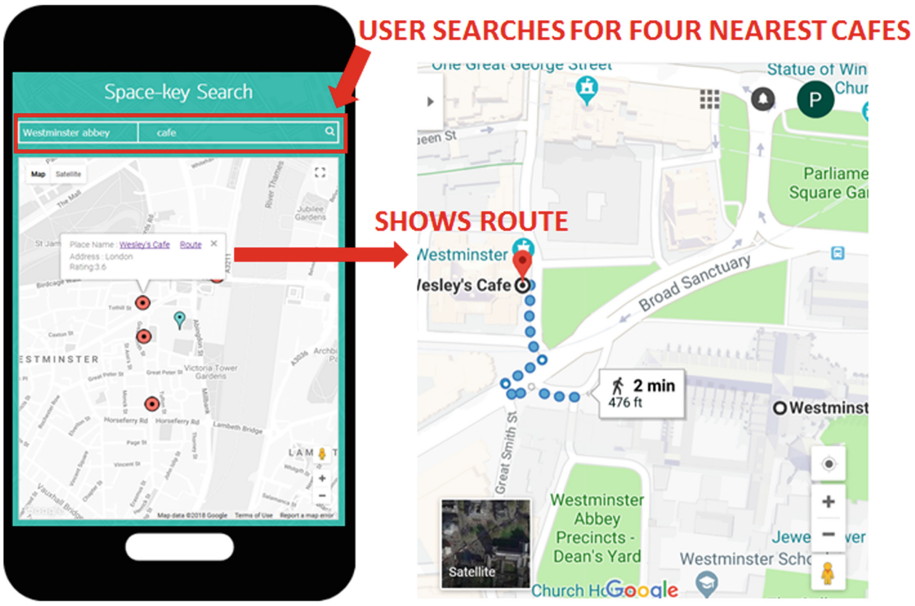
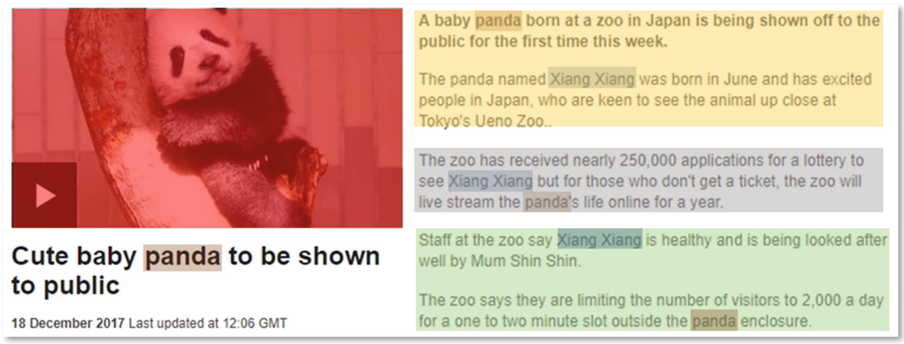


Fig. 9. Space-key search application for novice users[6].

the search results of the examples and could also freely modify the example operations.

Furthermore, another application of the map search which utilized our proposed query language (only using the "space-key" for a more simple and intuitive search) was also developed. 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␣␣␣␣cafe", to find the four nearest cafes to them on auto adjust map zooming (see Fig. 9). 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[5] (in Japanese). A mobile Web version of this application[6] was also developed for evaluation purposes.

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



Cute baby panda to be shown to public

18 December 2017 Last updated at 12:06 GMT

A baby panda born at a zoo in Japan is being shown off to the public for the first time this week.

The panda named Xiang Xiang was born in June and has excited people in Japan, who are keen to see the animal up close at Tokyo's Ueno Zoo..

The zoo has received nearly 250,000 applications for a lottery to see Xiang Xiang but for those who don't get a ticket, the zoo will live stream the panda's life online for a year.

Staff at the zoo say Xiang Xiang is healthy and is being looked after well by Mum Shin Shin.

The zoo says they are limiting the number of visitors to 2,000 a day for a one to two minute slot outside the panda enclosure.

Fig. 10. Example of web document search (Color figure online).

5.2 Applications of Web Document Search

Figure 10 shows an example of a web document search by the keywords "Xiang Xiang" and "panda". The search results are shown in the yellow portion, the gray portion, and the green portion. The yellow portion is the result of the search query: Xiang Xiang␣^panda, which is used to search all the sentences between "Xiang Xiang" and "panda" where "panda" precedes "Xiang Xiang" (see Fig. 10(b)). The grey portion and the green portion are the results of the search query: panda␣^Xiang Xiang, which is used to search all the sentences between "panda" and "Xiang Xiang" where "Xiang Xiang" precedes "panda" (see Fig. 10(c)(d)). The green portion is the result of the query: panda␣#<4lines␣Xiang Xiang, which is used to search for all the sentences including the keyword "Xiang Xiang" are before 4 lines from the keyword "panda" (see Fig. 10(d)). The picture shown in the red color is the result of the search query: panda␣#<1line␣.wmv, which is used to search a .wmv file is before 1 line from the keyword "panda" (see Fig. 10(a)).

6 Preliminary User Study

To evaluate the potential usefulness of our proposed query language, a user evaluation study was carried out in which 15 students from a university-level computer science course were recruited and asked to carry out a series of location search tasks. Participants had previous experience in using Google Maps in their daily life and were not familiar with the places which were used in the search assignments. Overall, the main aim was to determine whether such an operation based query language would be feasible for developers to learn and use. 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 condition) to complete 5 search assignments.

Each search assignment consisted of a task to search for places (e.g., pizza parlor) 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 500 m of Times Square. In another task, participants were asked to find the number of pizza parlors located within 700 m of Times Square which is also located 1000 m from the 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 participants spent on each task. To measure subjective user experience, the System Usability Scale (SUS) was used [2], 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 Google map 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 query language search system could indeed be useful.

7 Conclusion

In this paper, we proposed a novel spatio-temporal query language for spatial and temporal search which can be used to express complex search queries in a text equation format. Also, we use "space characters" (the space-key) between keywords which are used to express the geographical distance or time-length between matching objects in an easy and intuitive way for general users. We implemented a prototype of our proposed system as a RESTFUL Web service and developed some applications of map search and web document search to showcase how the query language could be used. We also experimented with Google Maps for map search and the results from our user study highlighted the potential usefulness of our proposed query language.

In the future, we look to expand our spatio-temporal query language to other search domains such as image or video search. Although, we have shown how our proposed query language could be used in map search and web document search, our proposed query language could easily be applied to spatial and temporal search within images and videos as well. For example, a query searching for video frames within a video that contains the object "B" and are within 10 s from the object "A" is expressed by "A␣10sec␣B".

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

References

1. Spatial Databases with Application to GIS. Morgan Kaufmann Publishers Inc., San Francisco (2002)
2. Brooke, J.: SUS-a quick and dirty usability scale. Usability Eval. Ind. **189**(194), 4–7 (1996)
3. Duckham, M., Kulik, L.: "Simplest" paths: automated route selection for navigation. In: Kuhn, W., Worboys, M.F., Timpf, S. (eds.) COSIT 2003. LNCS, vol. 2825, pp. 169–185. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39923-0_12
4. Elsevier: Elsevier R&D Solution: How can I search literature with reduced noise? Utilization of "proximity operator" in ScienceDirect & Scopus, 19 August 2015. http://jp.elsevier.com/__data/assets/pdf_file/0017/263240/Tips_Scopus_201508.pdf
5. Fu, K., Lu, Y.C., Lu, C.T.: Treads: a safe route recommender using social media mining and text summarization. In: Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 557–560. ACM (2014)
6. Güting, R.H., Schneider, M.: Moving Objects Databases. Elsevier, Amsterdam (2005)
7. Hwang, E., Subrahmanian, V.: Querying video libraries. J. Vis. Commun. Image Represent. **7**(1), 44–60 (1996)
8. Khosla, R.: Oracle text reference: 11*g* release 2 (11.2), b61357, 06 October 2015. https://docs.oracle.com/cd/E16338_01/text.112/b61357/title.htm
9. Kim, J., Cha, M., Sandholm, T.: SocRoutes: safe routes based on tweet sentiments. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 179–182. ACM (2014)
10. Liang, H.W., Hwang, Y.H.: Mobile phone use behaviors and postures on public transportation systems. PLoS ONE **11**(2), e0148419 (2016)
11. Lim, K.H., Chan, J., Leckie, C., Karunasekera, S.: Personalized tour recommendation based on user interests and points of interest visit durations. In: Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI 2015, pp. 1778–1784. AAAI Press (2015). http://dl.acm.org/citation.cfm?id=2832415.2832496
12. Mata, F., et al.: A mobile information system based on crowd-sensed and official crime data for finding safe routes: a case study of Mexico City. Mob. Inf. Syst. **2016**, 1–11 (2016)
13. Pradhan, S., Sogo, T., Tajima, K., Tanaka, K.: A new algebraic approach to retrieve meaningful video intervals from fragmentarily indexed video shots. In: Arisawa, H., Catarci, T. (eds.) Advances in Visual Information Management. ITIFIP, vol. 40, pp. 11–30. Springer, Boston, MA (2000). https://doi.org/10.1007/978-0-387-35504-7_2
14. Pradhan, S., Tajima, K., Tanaka, K.: A query model to synthesize answer intervals from indexed video units. IEEE Trans. Knowl. Data Eng. **13**(5), 824–838 (2001)
15. Quercia, D., Aiello, L.M., Schifanella, R., Davies, A.: The digital life of walkable streets. In: International World Wide Web Conferences, pp. 875–884 (2015)
16. Quercia, D., Schifanella, R., Aiello, L.M.: The shortest path to happiness: recommending beautiful, quiet, and happy routes in the city. In: Proceedings of the 25th ACM Conference on Hypertext and Social Media, pp. 116–125. ACM (2014)
17. Su, H., Zheng, K., Huang, J., Jeung, H., Chen, L., Zhou, X.: CrowdPlanner: a crowd-based route recommendation system. In: 2014 IEEE 30th International Conference On Data Engineering (ICDE), pp. 1144–1155. IEEE (2014)

18. Sujeet, P., Tajima, K., Tanaka, K.: Interval glue operations and answer filtering for video data retrieval. IPSJ Trans. Databases (TOD) **40**(3), 80–90 (1999). https://ci.nii.ac.jp/naid/110002724879/en/
19. Wakamiya, S., Kawasaki, H., Kawai, Y., Jatowt, A., Aramaki, E., Akiyama, T.: Lets not stare at smartphones while walking: memorable route recommendation by detecting effective landmarks. In: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp 2016, pp. 1136–1146. ACM, New York (2016). https://doi.org/10.1145/2971648.2971758
20. Weiss, R., Duda, A., Gifford, D.K.: Composition and search with a video algebra. IEEE Multimed. **2**(1), 12–25 (1995)
21. Yin, H., Sun, Y., Cui, B., Hu, Z., Chen, L.: LCARS: a location-content-aware recommender system. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 221–229. ACM (2013)