Parameterized Spatial SQL Translation for Geographic Question Answering

[1,2]Wei Chen
[1]Dept. of Computer Science and Engineering, [2]Dept. of Geography
The Ohio State University, Columbus, OH USA

***Abstract*—Spatial SQL (structured query language) is a powerful tool for systematically solving geographic problems; however, it has not been widely applied to the problem of geographic question answering. This paper introduces a parameterized approach to translate natural language geographic questions into spatial SQLs. In particular, three types of complexity are introduced and initial solutions are proposed to deal with these complexities. The entire parameterization process is implemented to generate spatial SQL templates for five types of geographic questions. It is suggested that our approach is useful for solving natural geographic problems using spatial functions such as those in a GIS.**

Keywords: spatial query, GIS, spatial ontology, machine learning, natural language processing, spatial artificial intelligence

I. INTRODUCTION

Geographic questions are common in everyday life. It would be useful to develop a GeoQA (geographic question answering) system that can automatically give exact answers to geographic questions. However, the task is challenging for several reasons. First, it is not trivial to develop a classification method that can effectively split questions into answerable types. Most work to date classifies questions based on expected answer types [1] rather than ways of answering these questions [2]. The latter is critically important for extending the capabilities of a GeoQA system [2].

Second, natural language to SQL translation has been previously studied including the lexicalist approach [3] evolutionary method [4]and tree-based machine learning approach [5]. However, the translation from geographic natural language to spatial SQL lacks research efforts. This is partially because of the complexities in processing natural language spatial relations and modeling spatial objects in metric and topological space [6, 7].

Third, traditional information retrieval (IR)-based QA systems can successfully answer different types of questions through data intensive approach [8] but are not as efficient as human intelligence. An IR-based system searches answers by pattern matching [9]; as a result, they unavoidably lead to certain level of redundancy since the same answer may be encoded multiple times in various forms across documents and all these documents have to be scanned in preparation of the best answer [10]. By contrast, a GIS-based GeoQA system utilizes knowledge of modeling and analyzing spatial data to solve problems using analytical routines that may not be readily available elsewhere.

This paper aims to develop a new approach for translating natural language questions to spatial SQLs through a parameterization process. We reveal three aspects of complexity during this process and discuss what decisions may be made to permit a valid translation. Parameters are defined and later used to populate a spatial SQL template. We showcase the process of translating five types of geographic questions into parameterized spatial queries. Finally, we demonstrated how our approach can be extended to answer more complicated questions by expanding an existing SQL.

II. PARAMETERIZATION OF SPATIAL SQL TRANSLATION

In a previous paper [2], we proposed a framework for solving geographic question answering problems using natural language processing, machine learning, ontologies and GIS. Here, we only discuss the SQL generation step as part of the entire geographic question answering process discussed in the previous paper. In this section, several complexities of the parameterization process is introduced.

A. Three types of complexity

1) Complexity I: same question, different ways to answer

Consider the following *location* question: *Where is Columbus, OH?* and possible ways to answer it in SQL. For discussion convenience, we adopt the syntax of PostGIS for constructing spatial SQLs. PostGIS is a spatial extension to the open source relational database Postgres for handling spatial data [11]. Further, we assume the following relation structure in a Postgres table called *Ohio*.

Table I. OHIO TABLE

Column Name	Row Value (one example record)
Name	Columbus
Type	City
LatLon	39.9833°N, 82.9833°W
Point	point blob
LocationDescription	In the middle of Ohio
Population	809,798

There are at least three ways for answering this question in spatial SQL.

SQL query I: return a latitude longitude string literal:

```
SELECT latlon
FROM Ohio
WHERE name='Columbus'
```

This will return the string 39.9833°N, 82.9833°W.

978-1-4799-4003-5/14 $31.00 © 2014 IEEE
DOI 10.1109/ICSC.2014.44

SQL query II: return a pair of x, y coordinates from a point-based geometry object. ST_X and ST_Y are spatial functions to obtain x, y coordinates respectively (see Table IV)

```
SELECT ST_X(Point), ST_Y(Point)
FROM Ohio
WHERE name='Columbus'
```

This will return -82.9833, 39.9833.

SQL query III: return a relative location

```
SELECT LocationDescription
FROM Ohio
WHERE name='Columbus'
```

This will return the string *In the middle of Ohio*, which in some cases is also an acceptable answer.

Since all above answers are generally acceptable, the choice of the best answer is subject to specific applications. This is largely decided by the need of the application and the level of granularity it requires for the results. Here, we argue that as long as a SQL can generate an acceptable answer, its template can be parameterized. Therefore, any of these above templates are parameterizable.

2) Complexity II: one object, multiple representations

The second type of complexity lies in the fact that the same spatial objects can be represented in various geographic formats. These formats are often required by a GIS to carry out analysis on these objects. The two most important formats of managing spatial data are rasters and vectors. Rasters use grids to represent a field view of objects while vectors model objects as points, lines and polygons. This duality of representation has long dominated GIS and spatial database based applications [12].

Vector-based spatial functions exist long before their raster-based counterparts[11]. As the vector view is more commonly adopted for modeling spatial objects, we only consider vector-based representation of space in this paper. Although most spatial objects can be represented as simple as abstract points, lines or polygons. Multi-points (i.e. a collection of points) or geometric collection (i.e. a collection of points, lines or polygons) may be needed such as in the case of representing *islands*, which is more suitable to be viewed as a multi-polygon object. Here, we only consider simple geometric representations of space, the point representations, to illustrate our parameterization approach.

3) Complexity III: one object, many ways of object-relational mapping (OEM)

The third type of complexity is potentially multiple ways of mapping spatial objects onto records in a relational database. This is largely a result of many ad hoc design choices in relational database applications. In our case, cities are saved as rows in a relation called *Ohio*, which is the name of the state that contains these cities. All city related attributes are saved in this state table. This scheme is convenient for answering questions about *Ohio* cities. But if a question concerns cities outside of Ohio, the program should direct data searching to other tables. This requires the development of spatial ontologies that specify the relationship between the hierarchy of spatial entities and the scheme information of a database.

In applications that all cities are stored in a large table, SQL queries should be reconstructed to adapt this design choice, for example, by adding a state='Ohio' criterion in the where clause. For simplicity, we decide not to provide a detailed discussion on object-relational mapping using ontologies of space in this paper. We choose only to use cities in Ohio as an example to illustrate our parameterization approach.

B. Parameterization process

Once decisions regarding the above three types of complexity have been made, we need to create SQL templates for answering questions. Here, we choose an inductive generalization approach to parameterize an SQL template. That is, we observe a sufficient number of questions of the same type (e.g. location question) and extract parameters that are shared across all these questions. We will discuss experiment data in the next section. Consider the following ways of asking the *location of Columbus*.

(1) Where is Columbus?
(2) What's the coordinates of the city of Columbus?
(3) What's the location of Columbus?
(4) Where is the capital of Ohio located?

To answer these questions, traditional IR-based system relies entirely on the similarity between sentences that contain answers and the lexical pattern of the question. Thus, all answers must be encoded somewhere in existing documents beforehand in hopes that a match can be found during the search.

The GIS-based QA system, the one we choose to implement, works differently. It classifies all questions into the same category not because they have the same type of head words or question words but because they require the same procedure to find answers using a GIS. Therefore, a universal spatial SQL template is sufficient to answer all these questions. The only template that may be needed in this case is the following. The query will return the longitude and latitude coordinates as location of the queried object.

```
SELECT ST_X([geom]),ST_Y([geom])
FROM [table_name]
WHERE name=[entity_name]{ and type=[entity_type]}
```

Here, we define following notations to denote required and optional parameters in a spatial SQL template:

[]: required
{}: optional

As we can see from the above template, once we know the question concerns location, the most important information to know is the name of the entity. In the above template, we can see there are three required parameters for the location question: `geom` column, `table_name`, and `entity_name`. The `geom` parameter is determined at runtime by resolving the spatial representations of objects. For point objects, the `geom` should correspond to a point geometric column; for polygon objects, it should point to a polygon-based geometric column. For any objects that potentially have dual representations (such as city), its ontology information should help point the entity to

its appropriate representations. But in our case, we simplify the problem by setting it to be fixed on the point geometry column of the table.

The second parameter `table_name` is also decided at runtime. Once a spatial entity is extracted, the program should be able to know where to search for data related to the entity based on its object-relational mapping information. In our case, we fixed the data table to Ohio for simplicity. The third parameter `entity_name` is required and is the key information that will be extracted from a location question. The last parameter `entity_type` is optional as it may not always be available in a sentence and the sentence may still be valid (see example sentences (1) and (2)). The names of the entities are usually proper nouns that begin with capital letters. The type of the entity may take values like city, town, and village.

C. *Annotating sentences*

Once we created these parameters, we can annotate new sentences using annotations of these parameters. This is done after the sentence parsing and is the result of information retrieval process. Both have been discussed on our previous paper [2]. The following shows the results of annotating four example sentences.

(1) Where is $[\text{Columbus}]_{\text{entity}}$?
(2) What's the coordinates of the $(\text{city})_{\text{type}}$ of $[\text{Columbus}]_{\text{entity}}$?
(3) What's the location of $[\text{Columbus}]_{\text{entity}}$?
(4) Where is the $[\text{capital of Ohio}]_{\text{entity}}$ located?

With the knowledge of entity names and entity types, we can now plug extracted information directly into the SQL template for location question. Along with fixed values of the `geom` column and `table_name`, we can construct the spatial SQL for querying the database.

```
SELECT ST_X(geom),ST_Y(geom)
FROM Ohio
WHERE name='Columbus' and type='city'
```

It is now possible to answer questions such as: *Where is Cleveland?* and *Where is the city of Cincinnati?* However, it is necessary to develop a lexicon database so that nouns like cities and towns can be recognized as valid types. For terms that can't be recognized by our database, the current system will ignore it during parsing.

The database structure to answer location questions may be as simple as Table II. Essentially, only three columns of information are needed: name of the entity, type of the entity, and spatial representation of the entity. However, such a simple table with limited number of fields and values may answer many different types of questions including four additional types of questions that we will discuss later. Table II only shows two examples of database records and it expands rightwards.

Table II. SIMPLIFIED TABLE OHIO

Column Name	Row Samples		
Name	Columbus	Powell	……
Type	city	village	……
Geom	<geom blob>	<geom blob>	……

III. QUESTION DATA AND TYPES

As a continuation of previous research [2], we used the same set of data obtained in the paper. Essentially, it was a survey corpus of about 800 questions from which we generalized the five most common geographic question types: (a) location question, (b) relative location question, (c) distance question, (d) proximity entity question, and (e) proximity buffer question. Due to space limitations, we only choose four different sentences for each type to showcase the various ways of asking the same type of question. For the detailed data collection process , one may refer to [2].

In Table III, questions are indexed by using a combination of type letter (a,b,c,d,e) and sequence index (1,2,3,4). For example, a1 means first example question of type a.

Table III. EXAMPLE QUESTIONS

Index	Example Question
a1	Where is Columbus?
a2	Where is Columbus located?
a3	What are the coordinates of Columbus?
a4	The location of Columbus?
b1	Where is Columbus with respect to Cincinnati?
b2	Relative to Cincinnati where is Columbus?
b3	Which direction of Cincinnati is Columbus in?
b4	Where is Columbus with perspective to Cincinnati?
c1	What is the distance from Columbus to Cincinnati?
c2	What is the distance between Columbus and Cincinnati?
c3	How far is Columbus from Cincinnati?
c4	Distance from Columbus to Cincinnati?
d1	What are the 3 nearest cities from Columbus?
d2	Which are the three closest cities of Columbus?
d3	Which 21 cities are closest to Columbus?
d4	What twenty one cities are nearest to Columbus?
e1	What cities are within 20 miles from Columbus?
e2	Which cities are inside twenty kms of Columbus?
e3	Which cities are within a 20 miles radius of Columbus?
e4	Cities are within 60 miles of Columbus and 80 miles of Cincinnati?

All these questions are classified by the type of parameters and spatial functions needed to find a solution. Location question require retrieval of the coordinates of spatial objects. Relative location questions require calculation of the azimuth and distance. Distance questions require the calculation of distance. Proximity entity questions require the calculation of distance between pairs of objects and ordering them by distance. Proximity buffer questions require the creation of an intermediate buffer area and the use of the buffer and the spatial operator *within* to find entities that fall into the buffer. The classification accuracy is above 85% on average.

IV. SPATIAL SQL TEMPLATES

A. *Essential spatial functions for generating spatial SQL templates*

Using the parameterization process discussed in section II, we may develop SQL templates for additional types of questions, type b,c,d, and e. As spatial functions are important

to the construction of spatial SQL templates, we give references to spatial SQL functions that are necessary to solve such questions (Table IV). All these spatial functions are available in PostGIS, an open source spatial database which follows the OpenGIS Simple Features standard for SQL [11].

Table IV. SPATIAL FUNCTIONS IN POSTGIS [11]

Spatial Function	Description
float ST_X(geometry a_point)	Return the X coordinate of the point, or NULL if not available. Input must be a point.
float ST_Y(geometry a_point)	Return the Y coordinate of the point, or NULL if not available. Input must be a point.
float ST_Azimuth(geometry pointA, geometry pointB	Returns the angle in radians from the horizontal of the vector defined by pointA and pointB.
float ST_Distance(geometry g1, geometry g2)	Return the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units.
geometry ST_Transform(geometry g1, integer srid)	Returns a new geometry with its coordinates transformed to the SRID referenced by the integer parameter.
boolean ST_Within(geometry A, geometry B)	Returns true if the geometry A is completely inside geometry B.
geometry ST_Buffer(geometry g1, float radius_of_buffer)	Returns a geometry that represents all points whose distance from this Geometry is less than or equal to distance. Calculations are in the Spatial Reference System of this Geometry.

Parameters developed for each type of questions are shown below. Here, we fixed the definitions of the table and the geometry column so that they are no longer parameters for generating SQL templates; but they may be considered differently in other cases.

Table V. PARAMETERS AND SPATIAL FUNCTIONS USED IN EACH TEMPLATE

Question Type #	Parameters	Spatial Functions Used
a	entity names	ST_X, ST_Y
b	entity_name1, entity_name2	ST_Azimuth, ST_Distance, ST_Transform
c	entity_name1, entity_name2	ST_Distance, ST_Transform
d	entity_name, order, number	ST_Distance, ST_Transform
e	entity_name, entity_type, distance, number	ST_Buffer, ST_Within, ST_Transform

As we can see in the above table, the classification scheme is largely decided based on the differences between the parameters and spatial functions used for solving questions. The templates for solving each type of question is given in Table VI. Table VI shows the possible values a parameter can take. As we can see, for our example, these values are mainly strings, float , integers or system reserved keywords such as asc in SQL. For simplicity, we also drop the optional parameter *entity_type* for later discussion as we assume all entities are cities.

Table VI. VALUE RANGES OF PARAMETERS

Parameters	Domain
entity name	String
entity names	[String1, String2, …]
order	[asc, desc]
entity_type	[city, village, town, organization, facility, …]
distance	Float
number	Integer

B. Spatial SQL templates

```
SELECT ST_X(geom), ST_Y(geom)
FROM Ohio
WHERE name in ([entity names])
```

Here, we provide the spatial templates for solving each type of geographic questions based on previously discussed parameterization process. They can be used to answer questions in Table III.

```
SELECT ST_Azimuth(t_in1.geom, t_in2.geom),
ST_Distance(t_in1.geom, t_in2.geom)
FROM Ohio as t_in1 and Ohio as t_in2
WHERE t_in1.name = [entity_name1] and t_in2.name
= [entity_name2]
```

1) Type a tempalte

The inputs here could be a list of place names and the output will be an array of the coordinates of those places.

2) Type b template

The inputs here are two entity names. The return will be an azimuth degree and the distance between the two. The order of the inputs affects the azimuth calculation. t_in1 means

```
SELECT
ST_Distance(ST_Transform(t_in1.geom,2163),ST_Tran
sform(t_in2.geom,2163))
FROM Ohio as t_in1, Ohio as t_in2
WHERE t_in1.name = [entity_name1] and t_in2.name
= [entity_name2]
```

the input table 1 and t_in2 means input table 2. Since both inputs are from the same table in our case, it is necessary to differentiate them for query purposes.

3) Type c template

In our database, coordinates were originally saved in the point geometry column in latitude and longitude degree units. They must be converted to those in linear units in order to facilitate distance calculation. `ST_Transform` is the spatial function to convert the coordinates of a spatial objects. 2163 is the SRID of the US National Atlas Equal area projection which is implemented in linear unit of meters.

```
SELECT t_out.name
FROM Ohio as t_in, Ohio as t_out
WHERE t_in.name IN ([entity names]) AND
t_out.name NOT IN ([entity names])
ORDER BY
ST_Distance(ST_Transform(t_in.geom,2163),ST_Trans
form(t_out.geom,2163)) [order]
LIMIT [number]
```

4) Type d template

For proximity entity question, only a list of closest/farthest entity names will be returned. Entity names are names of input entities.

5) Type e template

In this example, first a buffer is created using the spatial function `ST_Buffer`. Then the buffer is applied as the second parameter of the function `ST_Within` with the first parameter being the queried spatial objects. The buffer radius of 100 miles is converted to 160934 in meters.

```
SELECT t_out.name
FROM Ohio as t_out, Ohio as t_in
WHERE t_in.name IN ([entity names]) AND
t_out.name NOT IN ([entity names]) AND
ST_Within(ST_Transform(t_out.geom,2163),
ST_Buffer(ST_Transform(t_in.geom,2163),
[distance]))
{ORDER BY
ST_Distance(ST_Transform(t_in.geom,2163),ST_Trans
form(t_out.geom,2163)) [order]}
{LIMIT [number]}
```

V. CONCLUSION

Geographic question answering (GeoQA) is a complicated process involving several components that must synergistically work together. These components facilitate natural language-based parsing of sentences, retrieving of information, machine learning-based question classification and ontological modeling of space. In this paper, we elaborated on the details of spatial SQL translation, which is one critical step of solving geographic questions using geographic information system approach. In particular, we found three types of complexity that must be addressed before a natural language question can be converted a spatial SQL.

Results showed that compared with traditional information retrieval (IR)-based approach, spatial SQL translation approach is efficient for answering many questions of the same type. This is because each type of questions only corresponds to one template which can be reused regardless of the varied lexical forms of constructing natural language questions. By comparison, IR-based approach relies heavily on searching the entire fact table in hopes that it may find a match during the search. It is suggested that spatial SQLs may be potentially viable for solving large number of geographic questions especially if the use of GIS is involved. We suggest continued research in this regard and more experiments be conducted on both spatial and computational aspects of this research.

ACKNOWLEDGMENT

The author would like to thank Dr. Yungui Huang, Dr. Eric Fosler-Lussier, Dr. Rajiv Ramnath, Dr. Ningchuan Xiao, Dr. Daniel Sui, Dr. Satyajeet Raje, Dr. Grey Evenson, Dr. Jeff Olson, and three anonymous reviewers for their valuable suggestions and comments. Presentation of the work was funded by the research data and computing center at the research institute of the Nationwide Children's Hospital.

REFERENCES

[1] D. Metzler and W. B. Croft, "Analysis of statistical question classification for fact-based questions," *Information Retrieval,* vol. 8, pp. 481-504, 2005.

[2] W. Chen, E. Fosler-Lussier, N. Xiao, S. Raje, R. Ramnath, and D. Sui, "A Synergistic Framework for Geographic Question Answering," in *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, 2013, pp. 94-99.

[3] N. P. Demers, "A Lexicalist Approach to Natural-Language Database Front-Ends," Citeseer, 1999.

[4] A. Afonso, L. Brito, and O. Vale, "An evolutionary method for natural language to SQL translation," in *Simulated Evolution and Learning*, ed: Springer, 2008, pp. 432-441.

[5] A. Giordani, "Mapping Natural Language into SQL in a NLIDB," in *Natural Language and Information Systems*, ed: Springer, 2008, pp. 367-371.

[6] M. J. Egenhofer, "Reasoning about binary topological relations," in *Advances in Spatial Databases*, 1991, pp. 141-160.

[7] A. R. B. Shariff, M. J. Egenhofer, and D. M. Mark, "Natural-language spatial relations between linear and areal objects: the topology and metric of English-language terms," *International journal of geographical information science,* vol. 12, pp. 215-245, 1998.

[8] S. Tellex, B. Katz, J. Lin, A. Fernandes, and G. Marton, "Quantitative evaluation of passage retrieval algorithms for question answering," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 2003, pp. 41-47.

[9] R. D. Burke, K. J. Hammond, V. Kulyukin, S. L. Lytinen, N. Tomuro, and S. Schoenberg, "Question answering from frequently asked question files: Experiences with the faq finder system," *AI magazine,* vol. 18, p. 57, 1997.

[10] C. L. Clarke, G. V. Cormack, and T. R. Lynam, "Exploiting redundancy in question answering," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, pp. 358-365.

[11] R. Obe and L. Hsu, *PostGIS in action*: Manning Publications Co., 2011.

[12] J. Mennis, "Generating Surface Models of Population Using Dasymetric Mapping*," *The Professional Geographer,* vol. 55, pp. 31-42, 2003.