A Geographic Visual Query Composer (GVQC) for Accessing Federal Databases

Diansheng Guo
GeoVISTA Center, Department of Geography, Pennsylvania State University
302 Walker Building, University Park, PA 16802, USA
Email: dguo@psu.edu,URL:http://www.geovista.psu.edu/grants/dg-qg/intro.html

Abstract

The paper introduces a query tool which allows users (either database experts or non-specialist users) to visually formulate and execute *spatial* queries directed to large federal databases that contain geospatial data. By manipulating a limited set of query modules visually, the user can *correctly* and *quickly* compose complex spatial (and non-spatial) queries with several mouse clicks. Both the underlying database model and the query language are transparent to users.

1 Introduction

Large volumes of federal data (together with their associated geospatial properties) are being collected and managed with database systems. Accessing these databases with command-line-based query languages is difficult, error-prone, and tedious, particularly for users without abundant expertise about the database model and the query language (Egenhofer 1997). To enable non-specialist users to easily access these federal data, an efficient, effective, and flexible query interface is critical. The work presented here builds on developments over the past decade in GIScience to facilitate interaction with GIS (Qian 1998; Egenhofer and Kuhn 1999; Qian 2000) and in database and application fields to develop visual query systems (VQS) that make it easier for users to compose queries (Catarci, Costabile et al. 1995).

The Geographic Visual Query Composer (GVQC) introduced here can be viewed as a tool that allows users to visually formulate and execute a query statement directed to federal databases containing geospatial data. By manipulating a limited set of query modules visually, the user can compose complex spatial (and non-spatial) queries with just several mouse clicks. The database schema is tightly integrated with each query module, which facilitates the correct configuration of queries. GVQC also allows users to get/edit text-based SQL statements, which are automatically extracted from the visual design. This feature is very useful for novice users who want to learn the query language as well as for expert users who want to edit a query once constructed. A visual query design (i.e., a visual query statement) can be saved and then loaded later. The current version of GVQC works seamlessly with Oracle spatial databases.

2 Design and Implementation of GVQC

The ultimate goal of GVQC is to help users efficiently formulate queries and retrieve data. To achieve this goal, two major parts of work are introduced here: (1) the decomposition of complex queries into simple query modules, and (2) the visual formulation and configuration of queries.

2.1 Taxonomy of Query Modules

GVQC views a complex query as a hierarchical composition of simple queries. GVQC consists of a finite set of query modules, each of which will either formulate a simple query or process input from simple queries to formulate a composite query. An *atomic query* is a query that cannot be decomposed into simpler ones. Those modules that formulate an atomic query are referred as "*atomic query modules*", while those that process input queries are referred as "*composite query modules*". According to the data type they address, query modules can also be grouped into four categories: attribute modules, Boolean modules, spatial modules, and temporal modules. Temporal modules are not yet implemented and, thus, not included in this paper. Figure 1 shows a complete taxonomy of query modules in GVQC.

Attribute query modules are non-spatial queries on alphanumerical data types. This group contains four modules: (**1**) *Numerical Attribute Module*—to formulate constraints on numerical data types (e.g., float, double, integer, etc.); (**2**) *Nominal Attribute Module*—to formulate constraints on alphabetic data types (e.g., char, varchar, etc.); (**3**) *Attribute Join Module*—join two queries based upon alphanumerical relationships between two columns from the two input queries; (**4**) *Attribute Selection Module*—to select desired attributes without any constraints, which is often combined with other modules to form queries that return a set of attributes based on constraints of other attributes.
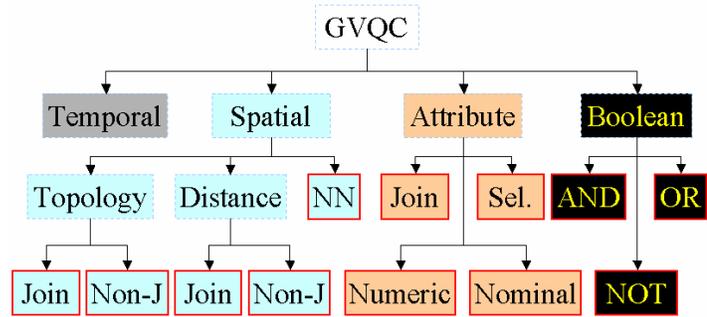

Figure 1: taxonomy of query modules in GVQC

Boolean query modules simply combine input queries with Boolean operators. This group contains three modules: (**1**) *AND Module*—combines two input queries with an "and" operator; (**2**) *OR* Module—combines two input queries with an "OR" operator; and (**3**) *NOT* Module—negates an input query with a "not" operator.

Spatial query modules are for spatial query formation, which include: (**1**) *Topological Join Module*—to join two input queries based on the topological relationship between the spatial data columns in each input query (see figure 2) (**2**) *Topological Non-Join Module*—to formulate a constraint on a single spatial column based on its topological relationship with a given spatial object (not a column), e.g. a point. For example, *find the county (from COUNTIES table) that contains point (longitude = -45.1, latitude = 50.0)*.

(**3**) *Within-Distance Join Module*—to join two input queries based on a metric relationship (within-distance) between the spatial data columns from each input query. For example, *find all pairs of city and lake (join table CITIES and LAKES) for which the city and the lake are within 1-mile distance*. (**4**) *Within-Distance Non-Join Module*— to formulate a constraint on a single spatial column based on its topological relationship with a given spatial object (not a column). For example, *find all highways (from table ROADS) that are within a 10-mile distance from my current location (long. = -77.85, lat. = 40.81)*. (**5**) *Nearest-Neighbor Module*—to query *N* nearest neighbors of a give spatial object. For example, *find 3 nearest cities (from table CITIES) to my current location (long. = -77.85, lat. = 40.81)*.

With the above twelve modules, complex spatial and non-spatial queries can be constructed in a visual manner, which is easy to understand, adjust, and modify.
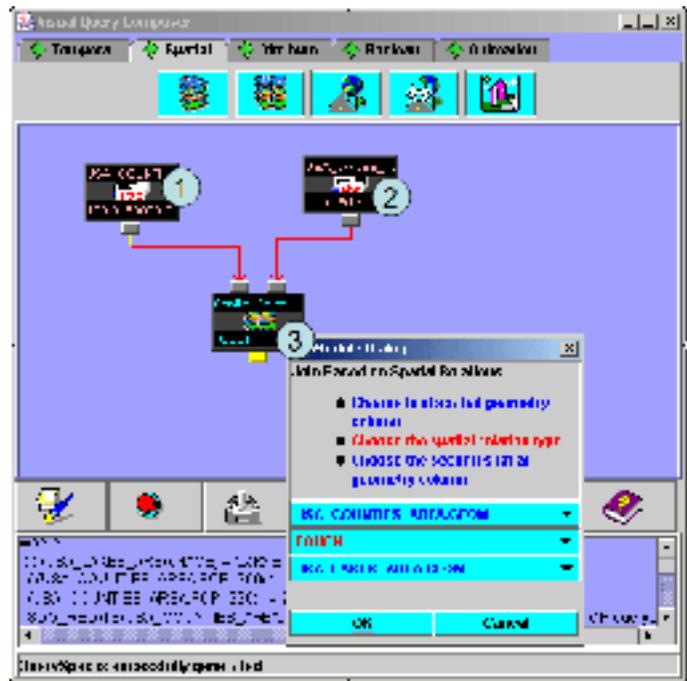

Figure 2: the overall system interface. (1) A numerical attribute module for querying COUNTIES.pop1990. (2) A nominal attribute module for querying LAKES.name. (3) A spatial topological join module for querying those counties from *(1)* that border any lake from *(2)*.

2.2 Formulating and Configuring Queries

A composite query module can automatically extract schema information from the input query modules. As in figure 2, the spatial relational (topological) join module gets inputs from a numeric range query module, which represents an atomic query "*select POP1990 from COUNTIES, where POP1990 between 1,000 and 50,000*", and a nominal query module, which represents an atomic query "*select NAME from LAKES where NAME='Erie'*". Then the spatial join module automatically recognizes those columns of spatial geometry type from both input queries, i.e., the GEOM column in COUNTIES and GEOM column in the LAKES tables. Thus, all the necessary schema information is already there and the user only needs to specify the topological relation (e.g. contains, touch, etc.)

The main interface for GVQC has five parts (figures 2 and 3). (1) Top-most is the tabbed *modules panel* that contain five groups of modules (here we are focusing on 3 groups, i.e., spatial, attribute and Boolean). (2) The large light-blue area is the *module arena*, which is the visual design area for formulating queries. (3) Further below is a collection of *system function buttons*. A set of buttons allow users to save, load, and clear visual designs, generate an SQL statement from a visual design, close the database connection and system, and obtain help. (4) Below the buttons is a text area for viewing and (optionally) editing SQL statements. (5) Dialogue boxes appear in the foreground. Each module has a dialog window to configure itself. In figure 2, the configuration dialog for the spatial join module is shown. A double-click on the module's icon in the arena will activate its configuration dialog.

Figure 3: an example query.

To formulate an atomic query, the user drags an atomic module onto the arena, double clicks it to open its configuration dialog, selects a table, selects a column, then either drags the slider bar to set a numerical range query or selects nominal values to set nominal constraints. To formulate a composite query, the user selects and configures several atomic query modules and connects them with one or more composite modules. To build the connection, the user simply drags the mouse from one module tab to another.

3 An Example Query

The example dataset covers the entire USA and contains 5 spatial data layers:

Figure 4: the query result

STATES, COUNTIES, CITIES, ROADS, and LAKES. Each layer has a spatial geometry column (storing points, lines, or polygons), and many other non-spatial data columns (demographic and other census variables). A simple query scenario is detailed below to illustrate the GVQC.

The scenario is that the user is interested in the relative impact of main highways on jobs, income, and the general economy of places for blacks and whites. They start by looking at the number of mobile homes, black, and white populations of small towns or small to medium cities. One example query from a set of similar ones might be: for all cities with total population < 50,000, within the 4-state region of PA-WV-MD-VA, and that are within 5 miles of a main highway—give me the city names, black population, white population, mobile homes of each city. The SQL (with Oracle spatial extensions) statement for this query is as follows:

> **select** *CITIES.NAME, CITIES.WHITE, CITIES.BLACK, CITIES.MOBILEHOME, CITIES.POP1990, ROADS.NAME, CITIES.STATE_NAME*
> **from** *ROADS, CITIES*
> **where** *((CITIES.STATE_NAME IN ('Maryland', 'Pennsylvania', 'Virginia', 'West Virginia') AND (CITIES.POP1990) <= 50000.0)   AND   SDO_WITHIN_DISTANCE(ROADS.GEOM, CITIES.GEOM, 'distance=5') = 'TRUE')*

With GVQC, the user can visually compose this query with just several mouse clicks without knowing either the SQL syntax or the detailed database schema (see figure 3 and figure 4). The query statement is *automatically* extracted from the visual design. Expert users can also edit the SQL statement before executing it. The design can be saved and re-loaded for subsequent use.

4   Conclusion

Advantages of this visual query system can be generalized as follows: (1) dynamic visualization of related database schema information to help the user explore the database and construct correct/accurate queries; (2) flexibility in and ease of forming complex spatial and/or non-spatial queries; (3) clear visualization of the semantic hierarchy of a complex query, which is useful for both forming and understanding such queries (this advantage can be seen by comparing the visual query design and the SQL statement in figure 3); (4) the ease with which a query can be modified; (5) the ability to extract text-based SQL statements that allow users to read/learn the query language; (6) the extensibility of the system for adding new modules and new visualization capability to each module.

References:
Catarci, T., M. Costabile, et al. (1995). "Visual Query Systems for Databases: A Survey." Journal of Visual Languages and Computing **8**: 215-260.
Egenhofer, M. and W. Kuhn (1999). Interacting with Geographic Information Systems. Geographical Information Systems: Principles, Techniques, Applications, and Management. D. Rhind, Wiley: New York**:** 401-412.
Egenhofer, M. J. (1997). "Query processing in spatial-query-by-sketch." Journal of Visual Languages and Computing **8**(4): 403-424.
Qian, L. (1998). Design of A Visual Query Language for GIS. The 8th International Symposium on Spatial Data Handling, Burnaby, B.C. Canada.
Qian, L. (2000). Visual Query Language. Department of Geography. State College, The Pennsylvania State University.