Interfacing the SAND Spatial Browser with FedStats Data[*]

Hanan Samet
Františsek Brabec
Gísli R. Hjaltason
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
{hjs,brabec,grh}@umiacs.umd.edu
www.cs.umd.edu/{~hjs,~brabec,~grh}

Abstract

FedStats enables ordinary citizens to access and search official statistics of numerous Federal agencies without knowing in advance which agency produced them. This is done via the internet through the world wide web. This data can be retrieved by topic, responsible agency, keywords, and press release. There is also a limited capability of retrieval of data by county or state. In particular, this is done by selecting the appropriate name from a menu or clicking on the desired area on the map. A proposal is made to use the capabilities of the SAND Spatial Browser to provide more power to users of FedStats. In addition, the issues involved in interfacing FedStats data with the SAND are discussed. Using the SAND Spatial Browser, instead of just being able to retrieve data for a particular location or a region with a predefined boundary, users can define the spatial region of interest with greater specificity. They can also make use of ranking which is the ability to retrieve data in the order of distance from other instances of the data or aggregates of data that are user-defined. The result includes the ability to perform clustering such as that provided by a Voronoi diagram.

1 Introduction

FedStats [1] enables ordinary citizens to access and search official statistics of numerous Federal agencies without knowing in advance which agency produced them. This is done via the internet through the world wide web. The data includes forecasts, projections, statistical tabulations, surveys, and other collected or derived data. Data can be retrieved by topic, responsible agency, keywords, and press release. There is also a limited capability of retrieval of data by county or state. In particular, this is done by selecting the appropriate name from a menu or clicking on the desired area on the map. In this paper we propose to use the capabilities of the SAND Spatial Browser to provide more power to users of FedStats. In particular, instead of just being able to retrieve data for a particular location or a region with a predefined boundary, users can define the

[*]This work was supported in part by the National Science Foundation under Grants EIA-99-00268, IIS-00-86162, and IRI-9712715.

spatial region of interest with greater specificity. They can also make use of ranking which is the ability to retrieve data in the order of distance from other instances of the data or aggregates of data that are user-defined. The result includes the ability to perform clustering such as that provided by a Voronoi diagram.

The rest of this paper is organized as follows. We first give an an overview of the SAND system in Section 2. Section 3 discusses the issues involved in interfacing FedStats data with SAND. Section 4 contains an example, while Section 5 contains some concluding remarks.

2 SAND

SAND is a GIS system developed at the University of Maryland to deal with spatial and nonspatial data [5]. It can handle two-dimensional data such as country boundaries, river paths, and city locations and facilitates the response to queries involving them such as finding the closest hazardous waste site to the border of a particular state. A major feature of SAND is the ability to find not just the closest hazardous waste site to a particular location or spatial object, but, instead, incrementally producing a list of all hazardous waste sites ordered by their distance from the particular location or spatial object.

In addition to aiding exploration of the data set by queries on spatial attributes, SAND also permits queries that involve the non-spatial attributes of the data. Taking our hazardous waste site example a bit further, we might have stored with each hazardous waste site some other attribute such as the level of pollutants or the nature of the dangerous chemicals that are present. Now, instead of just examining the hazardous waste sites that are near the Canadian border, we could just look at the ones that have at least a particular level of pollutants. Thus we see that SAND aids in the exploration of the data set. Presumably such a capability would be useful to scientists exploring other data as well.

SAND (denoting Spatial And Nonspatial Data) adopts an extended relational data model. As in traditional relational database systems (RDBMS), SAND organizes data into a collection of tables, where each table consists of a set of tuples (i.e., records), and each tuple is an aggregation of simple data types called attributes. Tables are handled in much the same way as disk files. In SAND, different table types allow tuples to be accessed in a number of ways (e.g., by number, or by contents), and these access methods are visible to the user who can use them directly. This is in contrast to a typical RDBMS which tries to hide such details from the user, providing him instead with the means to describe the data to be retrieved, and a query optimizer which can put the available (but hidden) access methods to the best use.

All tables in SAND respond to a minimal common set of operators, (e.g., **open**, **close**, get **first** tuple, get **next** tuple), while specialized table types offer additional capabilities. SAND currently defines three table types: relations, linear indices and spatial indices. Each table type supports an additional set of operators which is appropriate to its function in a database environment. The function of most of these operators is to alter the order in which tuples are retrieved and, by extension, the behavior of **first** and **next**.

Relations in SAND are tables which support direct access by tuple identifier (*tid*). Ordinarily, tuples are retrieved in order of increasing *tid*, but the operation **goto** *tid* can be used to jump to the tuple associated with the given *tid* (if it exists).

Linear indices are implemented as B-trees. Tuples in a linear index are always scanned in an order determined by their contents. The ordering between two tuples is decided by applying operator **compare** to their attribute values. Linear indices support the **find** operator, which retrieves from the disk storage the tuple most closely resembling a tuple value given as an argument. The **find** operator can also be used to perform range searches, i.e., all tuples corresponding to keys lying between two given values can be retrieved by **find**ing the tuple corresponding to the lowest bound and then using **compare** and **next** until the key exceeds the highest bound.

Spatial indices are implemented as PMR-quadtrees[7, 9]. The default scanning order of a spatial index is not determined by any particular constraint, but is designed to permit the full contents of the table to be examined as quickly as possible. Spatial indices support a variety of spatial search operators, such as **overlap** for searching tuples that intersect a given feature, or **within** for retrieving tuples in the proximity of a given feature. Spatial indices also support **ranking**, a special kind of access method whereby tuples are retrieved in order of distance to a given feature or set of features.

SAND implements attributes of common non-spatial types (integer and floating point numbers, fixed-length and variable-length strings) as well as two-dimensional geometric types (points, line segments, axes-aligned rectangles, polygons and regions). All attribute types support a common set of operators. Recall that non-spatial attribute types support the **compare** operator which is used to establish a total ordering among attribute values of the same type. No total ordering is possible among spatial values, but other kinds of relationships are supported by operators such as **distance** which computes the Euclidean distance between any two spatial values, or **intersect**, which determines whether two spatial values overlap.

The SAND kernel provides the basic functionality needed for storing and processing spatial and non-spatial data. In order to access the functionality of this kernel in a flexible way, we opted to provide an interface to it by means of an interpreted language. We chose *Tcl* [8] for that role, mainly because it offers the benefits of an interpreted language but still allows code written in a high-level compiled language (in our case, C++) to be incorporated via a very simple interface mechanism. Another advantage offered by Tcl is that it provides a seamless interface with *Tk*[8], a toolkit for developing graphical user interfaces.

The SAND Browser is our first attempt to provide a graphical user interface to the facilities of SAND. It permits the visualization of the data contained in a SAND relation by specifying two types of controls: the scan order in which tuples are to be retrieved and an arbitrary selection predicate. The tuples satisfying the query are obtained in an incremental order. Users rarely need to wait too long to get visual feedback provoked by an action.

The class of queries currently implemented in the SAND Browser is restricted to selections and spatial semijoins. The user specifies queries by choosing the desired selection conditions from a variety of menus and dialog boxes. Spatial values can either be drawn on the appropriate display pane or typed in by filling forms. Query results can either be displayed interactively using the *First* and *Next* buttons or saved in relations for use in subsequent SAND queries.

One of the key features of the SAND system is the support of the *ranking* operation. It enables finding objects in the order of their proximity to other objects. Ranking can be viewed as a spatial analog to sorting. For example, it is not possible to order a collection of points in the same sense that a collection of numbers can be sorted in ascending or descending order. However, it is possible to *rank* points in ascending or descending order of distance from a given point or spatial object.

In SAND, ranking is performed incrementally[6]. This means that once the parameters for a ranking operation are given, the first (and closest) tuple is returned as soon as possible. Thus the searching algorithm is executed just until the point where the first tuple in the ranking order can be established. Consecutive tuples are obtained in a similar fashion. The key point of the algorithm is that having computed the first k elements, to get the $(k+1)^{st}$ element we need not sort the $k+1$ elements. In other words, we resume the search for the $(k+1)^{st}$ element from the point at which we obtained the k^{th} element. This is a better solution than sorting the entire data set especially when we may only need a few of the closest elements rather than all of the elements in which case sorting the set would have been a good idea. This characteristic is extremely important in an interactive environment.

3 Interfacing FedStats data with SAND

The original design of SAND was as a standalone system. FedStats users would need remote access to SAND's functionality, and the best way to provide this is to make SAND available over the internet. We are in the process of developing a client-server version of the system that allows the actual database to be run in a central location maintained by spatial database experts while end users acquire a Java-based client piece that provides them with a gateway into the SAND spatial engine. Special provisions need to be made to enable end users to import their own data into the system. This also includes the ability to build indexes on it to facilitate efficient query responses. In this client-server scenario, all the data is stored on the server side in SAND's internal format so steps need to be taken to move the end user's dataset from their location to the server location and convert the data from the format in which it was gathered and/or delivered to SAND into a format that SAND can understand and use.

We have developed several conversion utilities that allow import of data from various external formats into SAND. One of the formats we are currently working on, and will be supporting in the near future, is Microsoft Excel. The spreadsheet format of Excel is fairly similar to SAND's native format and thus the conversion process can be made to appear simple to the end user by automating many interim steps.

Users familiar with spreadsheets, and Excel in particular, store the data in tables arranged in rows and columns. Typically, each row would represent an entry in the dataset, while columns represent individual attributes in each entry. SAND takes a similar approach so that the only step in the conversion process with which the user needs to be concerned is the specification of the meaning of each of the columns. In particular, SAND needs to know the data types of each column (e.g., a string in case of a name, integer in case of a city population, a pair of floating point values for coordinate values of a point, etc.). The user also needs to assign a name to each of these attributes to be used by SAND for referring to that attribute.

Once this assignment is done, the user needs to choose which attributes need to have indexes built on them. Typically, indexes would be built on spatial attributes, names, certain values, etc. Indexes probably do not need to be built on attributes that are purely descriptive and on which no searches would be performed. Examples of such attributes include columns in the original spreadsheet that were used to store comments, etc. If there is some doubt on the utility of some columns, there is no harm in building indexes on all columns (attributes) in the entry. Other more tricky examples include attributes whose values may be different from those used by the SAND system for support. For example, the names of states in the case of addresses are often two letter postal abbreviations while the internal specification of the states is done in part with their full textual names. This is an issue that needs to be resolved so that users can have a seamless integration with the spatial data provided by the SAND system (e.g., a map of the US for supporting queries that do not merely involved latitude and longitude specifications of locations).

We are developing a web-based utility that would guide the user through this process. The user would be asked to upload the Excel file (exported into a text file where fields are separated by tabs) to the server computer and then describe the individual attributes as outlined above. Once all the information is entered, the system will convert the text file into SAND's own internal format and build indexes on selected attributes. Finally, the data will be uploaded to a location that is accessible by the server portion of the SAND system.

Immediately, the user will be able to access the system remotely and start utilizing the data. In order to be able to support a large number of simultaneous uploads, we will make use of the BISTRO system with a set of pre-defined authorized servers that we call FedStats-Servers [4].

4 Example

As an example, we used two Excel files corresponding to EPA-regulated facilities that have Chlorine and Arsenic, respectively. For each file, we had the following information: EPA-ID, Name, Street, City, State, Zip Code, Latitude, Longitude, followed by flags to indicate if that facility is in the following EPA programs: Hazardous Waste, Wastewater Discharge, Air Emissions, Abandoned Toxic Waste Dump, and Active Toxic Release. Each of the programs was represented in the Excel file by a separate column and the appearance of the entry 'Y' indicates that the facility participates in the EPA program.

We converted this data to a SAND relation having the spatial attribute 'location' corresponding to the latitude and longitude, which was stored using a PMR quadtree for points. We added an attribute 'tuple-id' which serves to distinguish between the tuples in the relation thereby making them unique. There are several ways of implementing the nature of the program in which the facility participates.

1. Have an attribute called 'program' that contains the names of the programs in which the facility participates. The drawback of this solution is that the result is not in first normal form.

2. Make use of five Boolean attributes of the form 'is-program' to indicate if the facility participates in the program (e.g., 'is-hazardous-waste', etc.). The drawback of this solution is that users must formulate their queries in terms of this construct and its Boolean value which requires them to know how the Boolean value is specified. In particular, there are many ways of specifying Boolean values (e.g., '0' and '1', 'false' and 'true', 'no' and 'yes', etc.).

3. Have one tuple for each program in which the facility participates, and use the field 'program'. This is fine when each facility participates in just one or a few programs. The drawback is that in the case of queries on the basis of the values of other attributes, a facility will be retrieved as many times as the number of programs in which it participates. This is a classic problem in databases known as the duplicate problem [2, 3]. In our example, this drawback is alleviated by using the 'group by attribute-name' mechanism, which retrieves all tuples having the same value of the 'attribute-name' attribute simultaneously. In our example, we can use the 'EPA-ID' attribute. Note that it is not advisable to use the 'name' attribute in this case as there is no guarantee that two different EPA facilities with the same name do not exist.

We chose the third solution as it appeared to be the easiest to use and it did not require the user to know much about the underlying implementation.

Some queries that can be handled include:

1. Find all EPA-regulated facilities that have Arsenic that participate in the ”Air Emissions” program in states from Georgia to Illinois, alphabetically.

2. Find all EPA-regulated facilities that have Chlorine that participate in the ”Air Emissions” program that lie within the state of Arkansas or 30 miles within its border.

3. Find all EPA-regulated facilities that have Chlorine that participate in the ”Air Emissions” program that lie within 30 miles of the border of Arkansas (i.e., both inside and outside Arkansas).

4. For each EPA-regulated facility that has Arsenic, find all EPA-regulated facilities that have Chlorine that are closer to it than to any other EPA-regulated facility that has Arsenic. In order to avoid reporting a particular facility more than once, we use the ‘group by EPA-ID’ mechanism. Note that the result of this operation is analogous to a discrete Voronoi diagram where the sites are the EPA-regulated facilities that have Chlorine.

5. For each EPA-regulated facility that has Arsenic, find all EPA-regulated facilities that have Chlorine and that participate in the Air Emissions program that are closer to it than to any other EPA-regulated facility that has Arsenic. In order to avoid reporting a particular facility more than once, we use the ‘group by EPA-ID’ mechanism. Note that the result of this operation is analogous to a discrete Voronoi diagram where the sites are the EPA-regulated facilities that have Chlorine and that participate in the Air Emissions program.

5 Concluding Remarks

We have described a proposal to add a map browsing capability to FedStats making use of the SAND Spatial Browser. This is part of an ongoing project to develop spatial browsers and spreadsheets for use over several modes including on the web. The system is currently working on a small dataset. We are in the process of constructing a utility that would convert Federal government statistical data in Excel format to be compatible with the SAND Spatial Browser and making it usable to FedStats users.

References

[1] Fedstats: The gateway to statistics from over 100 U.S. federal agencies. `http://www.fedstats.gov/`, 2001.

[2] W. G. Aref and H. Samet. Uniquely reporting spatial objects: yet another operation for comparing spatial data structures. In *Proceedings of the Fifth International Symposium on Spatial Data Handling*, pages 178–189, Charleston, SC, August 1992.

[3] W. G. Aref and H. Samet. Hashing by proximity to process duplicates in spatial databases. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM)*, pages 347–354, Gaithersburg, MD, December 1994.

[4] W. C. Cheng, C.-F. Chou, L. Golubchik, S. Khuller, and H. Samet. Scalable data collection for internet-based digital government applications. In *Proceedings of the dg.o 2001 Conference: Connecting Government and the People Electronically*, Redondo Beach, CA, May 2001.

[5] C. Esperança and H. Samet. Experience with SAND/Tcl: a scripting tool for spatial databases. *Journal of Visual Languages and Computing*, 2000. To appear.

[6] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, June 1999. Also University of Maryland Computer Science TR-3919.

[7] R. C. Nelson and H. Samet. A consistent hierarchical representation for vector data. *Computer Graphics*, 20(4):197–206, August 1986. Also *Proceedings of the SIGGRAPH'86 Conference*, Dallas, August 1986.

[8] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[9] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.