An overview of the SAND Spatial Database System*

Claudio Esperança†
Gísli R. Hjaltason‡
Hanan Samet
Frantisek Brabec
Egemen Tanin
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742

December 22, 2001

Abstract

An overview is given of the SAND spatial database system, an environment for developing applications involving both spatial and non-spatial data. The SAND kernel implements a relational data model extended with several geometric functions and predicates as well as a spatial index. The main interface to SAND is through an embedded interpreted language. This permits the rapid prototyping of algorithms and makes SAND a useful tool both for applications and research. A graphical user interface that allows for easy database querying, and a client/server approach that simplifies remote access are also outlined.

*This work was supported in part by the National Science Foundation under Grants EIA-99-00268, IIS-00-86162, EIA-00-91474, and IRI-97-12715.

†Author's current address: COPPE, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Ilha do Fundão, CT, Bloco H, Rio de Janeiro, RJ 21949-900, Brazil.

‡Current address: RightOrder, Inc., 3850 N. First Street, San Jose, CA 95134.

1 Introduction

The dramatic rise in the use of the Internet and the worldwide web has led to a re-examination of traditional ways of looking at data. In particular, we increasingly find the need for applications to be location-aware. This means the incorporation of spatial data such as that found in maps. The first revolutionary step was the development of geographic information systems (GIS). In fact, one of the principal motivations for the development of GIS is to lessen the time necessary to produce a map. Maps have traditionally been formed as a result of a sequence of physical overlay operations and thus a reduction in the time necessary to perform such a task has always been viewed as desirable. Computerizing this physical process was viewed as such an improvement in speed that users were not overly concerned with making the computer execution time optimal. In particular, taking several hours to compute a result was considered acceptable in light of the time needed to tabulate the query manually and generate the output. Unfortunately, users today have been conditioned to get results quickly and do not want to spend much time waiting for them. In other words, they are willing to accept a result that is not 100% accurate and without so much detail provided that they can obtain it sufficiently quickly and with enough detail to make an intelligent decision (see [16] for a similar conclusion in a database environment).

1.1 Direct Manipulation

One of the principal reasons for this change of thinking is the familiarity of GIS users with spreadsheets. The invention of the spreadsheet is generally accepted as the most important factor in demonstrating the utility of the computer to the average user. It enabled her to answer questions such as "what if ..." without large commitments of time and money. One of the powers of the spreadsheet lies in its ability to make a database come alive in a visual sense. Moreover, the method of interaction with the spreadsheet is usually via direct manipulation (e.g., [29]) in the sense that users do not need to know how to write computer programs to get the output they desire. Instead, all operations are in terms of the basic entities and actions of the spreadsheet (i.e., the rows, columns, and drag-drop actions). Thus the spreadsheet enables users to have a very powerful decision support tool that responds to their requests instantly. They do not have to go to the printer to get the output to their queries. The output is in a format that facilitates subsequent queries.

Armed with their experiences with spreadsheets, it is not unreasonable for users to expect the same capability from their other decision making tools. Unfortunately, this is not so easy. For example, in the context of a GIS whose output is a paper map, once we have generated the paper map, we cannot simply annotate it with changes and pose subsequent queries on the modified map. Of course, we could save the output digitally and work directly on the screen. The limited resolution of the screen can be compensated by having a zoom capability (e.g., [5, 6, 21, 28]). Nevertheless, the fact that the output has the resolution usually expected for a paper map meant that the operations also took quite a long time to execute. This is especially troublesome when users do not necessarily require such resolution.

An additional problem for users is that frequently the type of queries that they wish to pose are a combination of spatial and nonspatial data. Spatial data is usually stored in a GIS (e.g., ArcInfo from ESRI) while nonspatial data is stored in a conventional database management system (DBMS) [14, 31, 33]. The drawback of most GIS is that they are usually very good for location-based queries such as "what feature is at location x?" However, they are not so good for feature-based queries such as "where is feature y?" [3]. Such queries are usually best handled by a conventional DBMS. Users want to answer both of these types of queries with equal ease. They do not want to know how the GIS or the nonspatial database are organized.

Handling such queries requires a seamless integration of spatial and nonspatial data [14, 31, 33]. The idea is to interact with a spatial database (GIS) in the same manner as we would interact with a nonspatial DBMS. Moreover, spatial operations should be executed using conventional database primitives. One problem with conventional DBMSs is that they are usually accessed via the aid of SQL (Structured Query Language) which

is rather cumbersome when it comes to nonstandard data such as maps and images [10], although there have been a number of attempts to adapt SQL to the spatial domain (e.g., [11, 13, 25]). Nevertheless, a graphical user interface is more appropriate as it enables users to query the underlying database without having to worry about whether or not a corresponding SQL extension has been defined already.

Conventional spreadsheets get their name from the fact that all the data is spread out in a tabular format and operations are specified in terms of combinations of rows and columns. An analogous problem-solving paradigm in a GIS is the overlay concept (e.g., [12, 32]). In this case, operations are specified as compositions of maps with the output of one or more operations serving as input to other operations. Frequently, in a GIS there is no need for the operation to run to completion to obtain the desired results. Often, we would like to proceed in a pipelined fashion where the first results of an operation are fed as inputs to subsequent operations. We characterize such a solution as being *incremental*. We use the term *browsing* (*browser*) to describe the physical process (processor) of (for) obtaining an answer incrementally.

1.2 Example

As an example of one of the composition queries that we wish to handle, consider "finding the closest county (in terms of distances in the plane) to Cook County (i.e., Chicago) with a bladder cancer mortality rate for white males greater than 7.5 per 100,000 people in the period 1970–1994 and a population greater than 1 million". What makes this query difficult is the presence of the spatial condition involving distances between two-dimensional regions. Conventional DBMSs facilitate retrieval on the basis of a particular attribute by building an index for it (i.e., sorting it). In the case of one-dimensional data like the mortality rate per 100,000 people and the population, this is quite simple as we have a zero reference point with which to sort the data. For example, assuming the existence of an index on the bladder cancer mortality rate per 100,000 people in the period 1970–1994, to find all counties in the order of the closeness of their mortality rate per 100,000 people to that of Cook County for which it is 8.5, we look up the value 8.5 in the index and then proceed in two directions along the index on the population attribute to obtain the nearest counties by mortality rate per 100,000 people in constant time. We do not have to rebuild the index if we want to be able to answer the next query which deals with the mortality rate per 100,000 people in Los Angeles county whose population is about 8 million.

Unfortunately, this strategy cannot be used when dealing with distances in the two-dimensional plane. For example, to find the county closest to Cook County, we could sort the counties according to their distances from Cook County. However, to find the closest county to Los Angeles County, the list of sorted distances from Cook County is not useful to us due to the non-additivity of distances in domains whose dimensionality is greater than one. In other words, the distance from Cook County to Los Angeles County is not equal to the sum of the distance from Los Angeles County to St. Louis County and the distance from St. Louis County to Cook County. Thus we need to be careful in the manner in which we represent the locations of the counties. In particular, we need to use an implicit spatial index that is based on spatial occupancy (e.g., a quadtree, R-tree, etc. [26, 27]) rather than an explicit spatial index that is based on distances from a particular reference point.

The query that we have just described is an instance of a process that we term *ranking*. Ranking is a byproduct of sorting. However, often, we are only interested in the first few values (e.g., the three closest counties to Cook County), in which case sorting the entire set of counties by distance from Cook County is an overkill. Moreover, as we saw, if we want the nearest three counties to Los Angeles County, then we must reinvoke the sort. Thus the most frequent solution is to calculate the k nearest counties to Cook County. The problem with this approach is that if we want the $k+1^{st}$ (e.g., the fourth) nearest county to Cook County, then we have to restart the computation and compute the $k+1$ (i.e., 4) nearest neighbors. Therefore, it is preferable to compute the nearest neighbors in an incremental fashion so that we need not compute more neighbors than are necessary. This is especially useful when we want to respond to queries such as finding

the nearest county to Cook County with a bladder cancer mortality rate for white males in the period 1970–1994 greater than 7.5 per 100,000 people and a population greater than 1 million since the nearest one may not satisfy the mortality rate and population conditions thereby necessitating finding the next nearest, etc.

1.3 Our System

SAND (denoting Spatial And Nonspatial Data) is a prototype spatial database system/GIS developed at the University of Maryland that has many of the above features. In particular, the SAND system contains a browser called the SAND Browser that enables the visual definition and execution of these queries. We are using the SAND system in digital government applications such as FedStats and the National Atlas of Cancer Mortality. The intended purpose of the SAND system is to be a research vehicle for work in spatial indexing, spatial algorithms, interactive spatial query interfaces, etc. The basic notion of SAND is to extend the traditional relational database paradigm by allowing table attributes to be *spatial* objects (e.g., line segments or polygons), and by allowing spatial indexes (such as quadtrees) to be built on such attributes, just as traditional indexes (like B-trees) are built on nonspatial attributes.

The rest of this paper is organized as follows. Section 2 describes the basic structure of the SAND system (i.e., the SAND kernel and the SAND interpreter), and which has recently been extended to function within a client/server environment. Section 3 presents the SAND Browser and the SAND Internet Browser, which provide a graphical user interface to the query facilities of the SAND system, as well as examples of their use in the context of digital government applications. Concluding remarks are drawn in Section 4 in addition to suggestions for future work.

2 SAND

SAND is divided into two main layers, the SAND kernel, and the SAND interpreter (see Figure 4). The SAND kernel was built in an object oriented fashion (using C++) and comprises a collection of classes (i.e., object types) and class hierarchies that encapsulate the various components. SAND adopts a data model inspired by the relational model. Thus, its core functionality is defined by the different types of tables and attributes it supports, and the class hierarchies that encapsulate this functionality are among the most important. Both of these aspects of the SAND kernel are defined in an extensible manner, so that new table and attribute types can readily be added to the system. The SAND interpreter provides a low-level procedural query interface to the functionality defined by the SAND kernel. Using the query interface provided by the SAND interpreter, we have built a number of useful tools. In addition to the interactive spatial query browsers described in Section 3, we have built a prototype for a high-level declarative query interface to SAND, modeled on SQL, and a prototype image database system [30].

The SAND kernel has many of the characteristics of full-featured relational database systems. For example, it has a block-based storage manager that caches blocks associated with the various tables (i.e., relations and indexes) in the system, with an LRU replacement policy. Furthermore, tuples (also termed *rows* and *records*) are laid out in blocks such that a block may contain multiple tuples, while large tuples may span multiple blocks. Nevertheless, due to main emphasis of our research, SAND does not currently support features such as transaction and concurrency support, or query planning and optimization.

2.1 Table Types

The table abstraction in SAND encapsulates what in conventional databases are known as relations and indexes. Tables are handled in much the same way as regular disk files, i.e., they have to be opened so that input and output to disk storage can take place. All open tables in SAND respond to a minimal common set of operators, such as **first**, **next**, **insert**, and **delete**. SAND currently defines three table types: relations,

linear indexes, and spatial indexes. Each table type supports an additional set of operators, specific to its functionality. The function of many of these operators is to alter the order in which tuples are retrieved, i.e., the behavior of **first** and **next**.

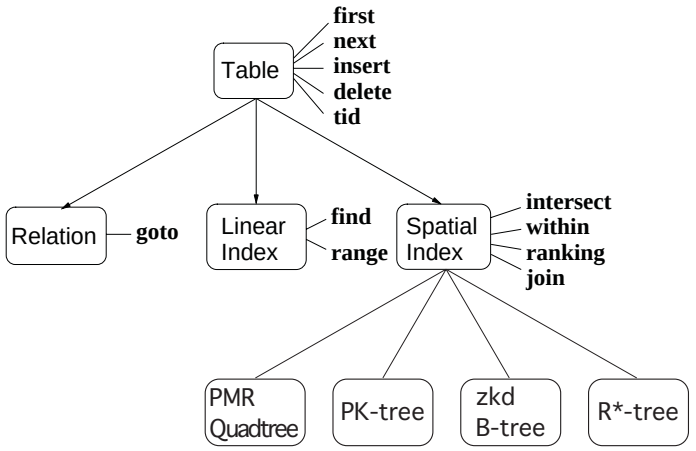


Figure 1: The class hierarchy of tables supported by SAND (the arrows denote class derivation).

Relations in SAND are tables that support direct access by tuple identifier (*tid*, which are composed of a block number and a tuple number). 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). For access by attribute values, indexes can be defined on attributes or groups of attributes of relations.

Linear indexes for non-spatial attributes are implemented using B-trees [8]. Tuples in a linear index are always scanned in an order determined by a linear ordering relation. Linear indexes support the **find** operator, that locates the first tuple in the index that is greater than or equal to the argument, and the **range** operator, that is used to perform range search.

Indexes can also be defined on spatial attributes in SAND. The SAND kernel defines an extensible framework for spatial indexes that makes it straight-forward to plug an already-implemented spatial access method into the system. Currently, the system supports indexing with the PMR quadtree [22], PK tree [34], zkd B-tree [23], and R*-tree [4, 15]. Spatial attribute values indexed by a spatial index may be represented in up to three ways (at least one of which must be supported by a given spatial index type): 1) *inline*, where the spatial attribute value is stored inside the spatial index, 2) *object table*, where a separate table is created to store copies of the attribute values, and 3) *from relation*, where the attribute values are accessed directly from the tuples of the indexed relation. The advantage of *inline* is that during search, all the information is present in the index, but it has the drawback that it makes the index larger. The advantage of *object table* over *from relation* is that the relation may contain many attributes, all of which would be accessed if the spatial attribute value of a tuple is needed during search. Moreover, the *object table* approach allows *clustering* the spatial attribute values in a way that optimizes I/Os (see [7]), without affecting the relation itself. Nevertheless, it has the drawback that the spatial attribute values are stored in two places, in the relation itself and in the object table.

A number of standard search operators are defined for spatial indexes, some or all of which may be implemented by a particular index type. These include **intersect**, for searching tuples that intersect a given feature; **within**, for retrieving tuples in the proximity of a given feature; and **ranking** [17, 19], for retrieving tuples in order of distance from a given feature (ranking is closely related to nearest neighbor queries). Furthermore, the **join** operator can be applied on two spatial indexes, where the join is either by intersection (i.e., a traditional spatial join) or by distance (termed *distance join* [18]).

1	2	2S	3	*N*
char	point	spoint	point3d	point(N)
char(N)	line	sline	line3d	box(N)
string	rectangle	lrectangle	box3d	
integer	polygon	spolygon	triangle3d	
float	region		triangleStrip3d	
boolean	icon			
array	image			

Figure 2: Attribute types defined for each dimension class.

2.2 Attribute Types

SAND implements attributes of common non-spatial types (integer and floating point numbers, fixed-length and variable-length strings, etc.) as well as various kinds of spatial types. Attribute types have an associated *dimension class*, that group together "compatible" attribute types, and attribute values have an associated dimensionality. The dimension classes currently defined in the system are labeled "1", "2", "2S", "3", and "*N*", where the numeric labels correspond with the dimensionality, "2S" denotes two-dimensional spherical geometry [2], and "*N*" denotes arbitrary dimensionality (i.e., the attribute types of that class support spatial objects of any fixed dimensionality). Thus, non-spatial attribute types are all in the class labeled "1", while each spatial attribute type is in one of the other classes. All the spatial classes (i.e., classes other than "1") contain at least an attribute type for points and another one for axis-aligned hyper-rectangles. The attribute types currently defined in each class are listed in Figure 2.

All attribute types support a common set of operations to convert their values to and from text, to copy values between attributes of compatible types, as well as to compare values for equality. Non-spatial attribute types also support the **compare** operator, which is used to establish a linear ordering between values of the same type. This is required so that non-spatial attributes can be used as keys in linear indexes. Spatial attribute types support a variety of geometric operations, including **intersect**, which tests whether two features intersect, **distance**, which returns the Euclidean distance between two features (used for the **ranking** operator), and **bbox**, which returns the smallest axis-aligned rectangle that contains a given feature (i.e., its minimum bounding rectangle). Some spatial types support additional operations. For instance, the *region* type supports operations like **expand**, which can be used to perform morphological operations such as contraction and expansion, and **transform**, which can be used in the computation of set-theoretic operations.

The attribute types listed in Figure 2 may be thought of as comprising a class hierarchy, with the base class "Attribute", and a derived base class for each dimension class, as partially depicted in Figure reffig-sandattr. However, for performance reasons and for increased flexibility, instead of relying on the object-oriented features of C++, we opted to develop our own attribute type manager that provides an extensible mechanism for attribute types and functions on them. The type manager maintains a registry of types, each of which has an associated string identifier (as listed in Figure 2) and a unique numeric identifier that is used internally. Furthermore, the type manager coordinates the creation and release of spatial objects, and the invocation of the common set of operations mentioned above. The type manager also maintains a function registry, where each function is also identified by a string, and may take an arbitrary number of arguments (each of which may be for a fixed attribute type or for an arbitrary one) and have a return value of several different types. The function registry is used for the spatial operations mentioned above, **intersect**, **distance**, and **bbox**, that are defined for all spatial attribute types, as well as for specialized operations that have been added as needs arose (e.g., **area** for computing the area of two-dimensional polygons). With the type manager it is easy to add new types and functions on them to the system, and the set of types and functions in the system is continually growing.

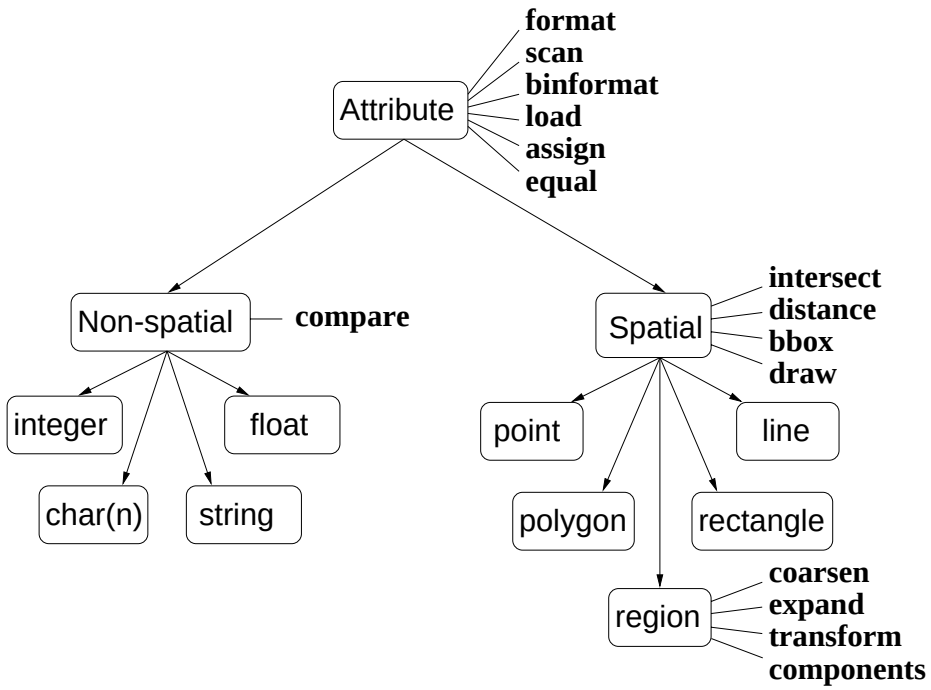


Figure 3: Some of the attributes types implemented in SAND and some of the operations defined on them.

2.3 The SAND/Tcl Interface

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 scripting language, *Tcl* [24]. Tcl 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* [24], a toolkit for developing graphical user interfaces.

The SAND interpreter provides commands that mirror all kernel operations mentioned in the previous sections. In some cases, a single command may cause more than one kernel operation to be performed. In addition, the interpreter implements data definition facilities. The processing of spatial queries is supported by interpreter commands that operate on spatial attributes and spatial indexes. While some of the commands available in the SAND interpreter are for accessing SAND kernel functionality, most are defined by the underlying Tcl interpreter and the Tk toolkit. In addition, we developed a Tk "widget" for displaying two-dimensional maps, used by the SAND Browser (see Section 3.1), that efficiently handles large data sets and provides zooming and panning facilities. Furthermore, system can be extended through Tcl's scripting capability by writing new methods or query strategies, which are either a standard addition or added by an application developer. In fact, the interpreter can be viewed as the unifying element of the whole SAND system (see Figure 4, which is a block diagram of the SAND system).

2.4 Client/Server Architecture

The SAND interpreter application (i.e., the executable) includes the SAND kernel codebase, since the interpreter directly accesses functionality of the kernel. Thus, any application built on top of the interpreter, such as the SAND Browser (see Section 3.1), runs in the same process as the SAND kernel, and displays maps on

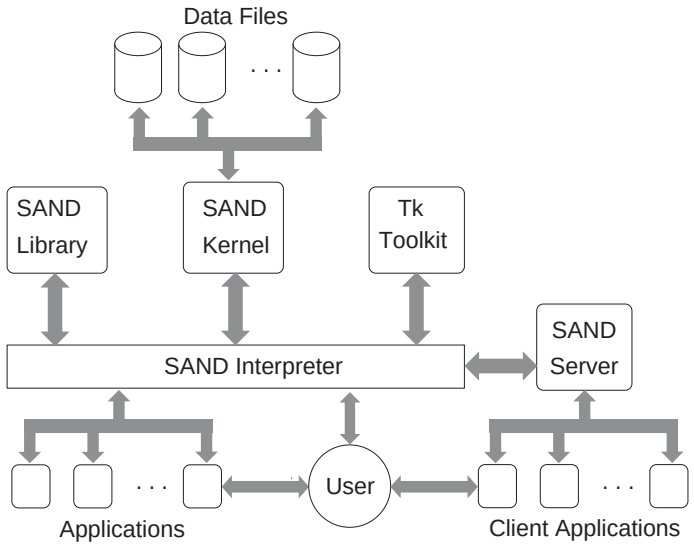


Figure 4: A block diagram of the SAND system.

the same computer[1]. Although for many uses, this is an adequate solution, the proliferation of the Internet makes it increasingly attractive to provide database access over a wide area network. To address this need, many web-based spatial data providers have adopted the approach of delivering maps as images, created by a database server (e.g., www.MapQuest.com and www.MapsOnUs.com). This an appropriate solution for many applications, and requires minimal resources for both hardware and software on the client side. Nevertheless, the resulting product has severe limitations in terms of available functionality and response time, and the transferred images do not have the same flexibility and representational power as the spatial data itself.

Based on the above considerations, we chose to adopt a client/server model where the actual data values (including those of the spatial data types) are transferred between the client and the server, and the client can issue queries to the server. One option would be to expose the full SAND/Tcl interface to the client, but this approach would invite a host of potential security risks. Thus, we designed a protocol that provides a more restricted access to the functionality of the SAND interpreter. The server itself is written in the SAND/Tcl script language, while we have developed a graphical client, written in Java, with functionality that resembles the SAND Browser (see Section 3.2). In an earlier effort, we also developed an OpenMap server interface for SAND, in cooperation with USGS [9].

3 Interactive Query Interfaces

In this section, we describe two ways of interacting with the SAND system in a graphical manner (Sections 3.1 and 3.2), and present examples of their use (Section 3.3).

3.1 SAND Browser

The SAND Browser provides 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.

[1]Strictly speaking, this is not true for systems that support the X windowing system. Nevertheless, relying on X is not a viable solution for delivering maps over the Internet.

The class of queries currently implemented in the SAND Browser is restricted to selections and spatial joins (e.g., distance joins and distance semijoins [18]). 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 in a manner somewhat analogous to a spreadsheet.

3.2 SAND Internet Browser

As mentioned in Section 2.4, the SAND Browser is not suitable for interactive map delivery over the Internet. In a more recent effort, we have developed another graphical query interface for SAND, that functions as a client to the server interface described in Section 2.4. This client interface, termed the SAND Internet Browser, is built using the popular Java technology, and is a relatively simple and lightweight application. Using Java provides platform independence while reducing installation and maintenance efforts. Like the SAND Browser, the SAND Internet Browser is more than a naive image viewer, but instead operates on vector data and allows the client to perform many operations such as zoom in/out or locational queries without communicating with the server. In essence, the client keeps a local *cache* of a portion of the whole database, which is only updated when additional or newer data is needed.

We see two different types of usages for our Java-based browser. First, the browser can be activated as an applet so that users across various platforms can access a spatial database on a remote machine without having to install the SAND system on their side. Second, the browser along with the SAND kernel can be installed on the client side. In the latter case, the browser can be utilized to view data from remote data sources, while frequently used data can be loaded to the local SAND database on demand, and subsequently accessed locally. In time, users can build up their own local databases and make it available over the network.

3.3 Sample Queries

Figure 5 is a screenshot of what a user interaction with the SAND Internet Browser might look like. It shows the relation corresponding to mortality rates per 100,000 for bladder cancer for white males for the time period 1970–1994. We have also overlaid it with the result of a clustering-like operation that is available in SAND. In particular, we have shown a partition of the underlying space with respect to the 17 counties with the highest mortality rates so that each county in each partition is closer to the county with the high rate in the same partition than to any other county with a high rate. The green dots indicate locations of high chlorine emissions obtained from the FedStats [1] website. The goal is to see if there is some spatial correlation between counties with a high incidence of bladder cancer and large chlorine emissions. As can be seen, locations with a large amount of chlorine emissions are not clustered around these counties. Thus these two events do not seem to be spatially correlated.

The scenario depicted in Figure 5 is analogous to a discrete Voronoi diagram and is a form of clustering. This clustering operation is available in both the SAND Browser and the SAND Internet Browser and can be achieved by executing an incremental *distance semi-join* [18] operation where the input relation corresponding to the high chlorine emissions map is joined with the high incidence of bladder cancer map and the join condition is based on proximity with the closest tuple pairs from the two sets being retained. Once the closest emissions-cancer pair (a,b) has been found, the next closest pair is found from the set of emissions tuples which excludes tuple a from participating. This process is continued until the closest high incidence of bladder cancer county has been found for each of the high chlorine emissions locations.

Figure 6 illustrates another sample query. In this query, nuclear facilities around a certain monitoring station along the northeastern U.S. and the Canadian border (Figure 6a) are computed in the order of their distance to this station. First, we define our query by selecting the location of the station and then the ranking

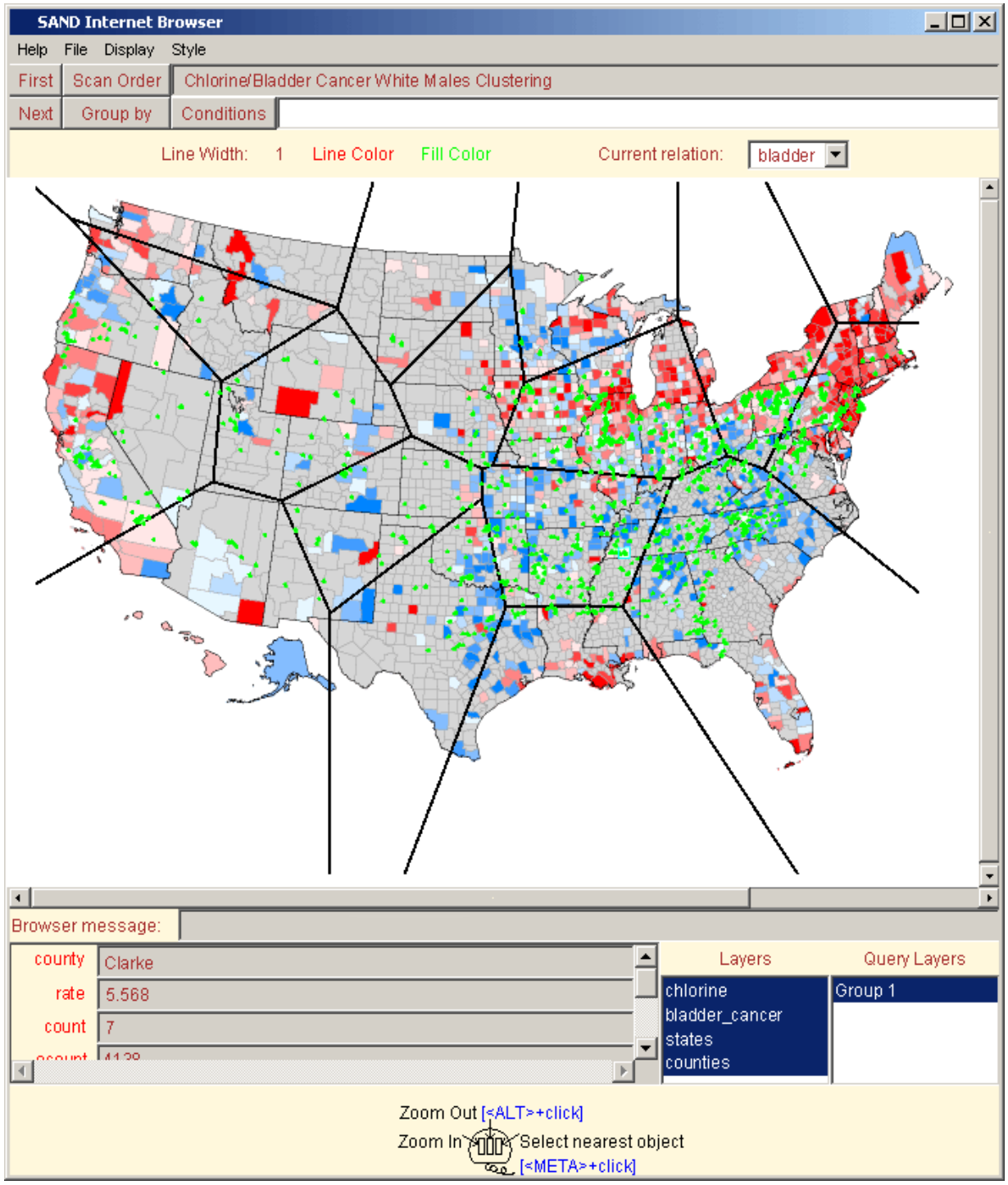
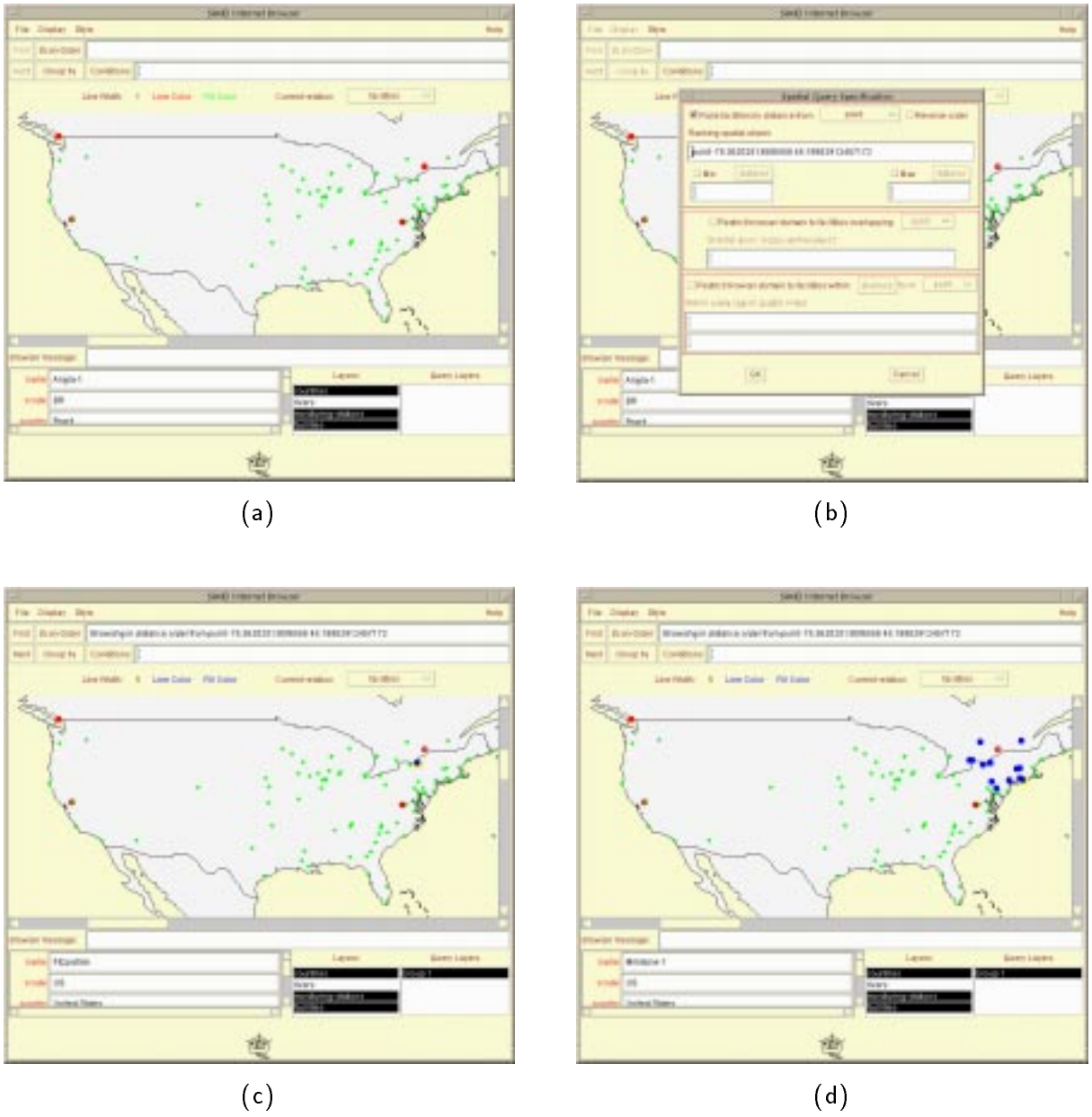


Figure 5: Sample screenshot of a possible user interaction with the SAND Internet Browser. The relation being displayed corresponds to classes of mortality rates per 100,000 for bladder cancer for white males for the time period 1970–1994. It is also overlaid with the result of a partition of the underlying space with respect to the 17 counties with the highest mortality rates so that each county in each partition is closer to the county with the high rate in the same partition than to any other county with a high rate. The green dots indicate locations of high chlorine emissions.

operation starts by displaying our first hit (Figures 6b and 6c). By clicking the "Next" button we can continue this operation as long as we want (Figure 6d). Again, if desired, the nuclear facilities relation can be partitioned with respect to the monitoring stations relation with a discrete Voronoi diagram (Figure 7).



(a) (b) (c) (d)

Figure 6: The nuclear facilities around a certain monitoring station along the northeastern U.S. and the Canadian border are computed. The green dots indicate nuclear facilities, the red dots indicate monitoring stations, and the blue dots indicate hits to our query. (a) Displays the two relations, monitoring stations and nuclear facilities; (b) the location of a certain station is chosen for a ranking query by distance; (c) the closest facility is displayed; (d) the query continues with other hits, incrementally.

Figure 7: The discrete Voronoi diagram, partitioning the nuclear facilities relation with respect to the monitoring stations relation, is depicted.

4 Concluding Remarks

SAND is an on-going project. At present, we are focusing on the client/server environment for which our efforts are in two directions. The first is on developing efficient caching methods that would balance limited client resources on one side and significant latency of the client/server link on the other while the low bandwidth of this link would be a concern in both cases. The second is to help users that want to manipulate data for prolonged periods of time by developing a peer-to-peer approach to provide them with the ability to download fairly large amounts of data more efficiently by better utilizing the distributed network bandwidth. In addition, we want to use the same mechanism to help them upload the results of their work to a remote server if needed.

The classic client/server approach for transferring data between the two ends of a connection assumes a designated role for each of the ends (i.e., a client + a server). It also ignores the fact that the needs of the two ends may be time dependent (i.e., congested periods of usage for the server). A pure peer-to-peer approach where the two ends/peers can assume both the roles of a client and a server from time to time may improve the overall network performance by resolving the congested situations. At a given time the server may be busy serving other requests (forming a congestion). The common solution to this problem in the realm of databases is to cache or forward results/requests. The novelty of the peer-to-peer approach is converting the static configuration of forwarding requests to a highly dynamic one where a persistent storage is formed from a pool of clients and servers (peers). A request to download/upload data can be performed by a set of selected peers from this pool at a given time that optimizes the network performance. Keeping alive and fresh copies of the data (and hence a directory of active peers) forms a challenging research problem in this area. Hybrid configurations where a main server (e.g., a government/company operated server) exists are also possible.

Other work includes adding a map browsing capability to FedStats using the SAND Browser. This work also involves the construction of a utility that would convert Federal government statistical data in EXCEL format to be compatible with the SAND Browser. In addition, work is ongoing to further develop the concept of a spatial spreadsheet [20] using the SAND Browser. It is interesting to note that SAND has already been used for a prototype image database system [30].

References

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

[2] H. Alborzi and H. Samet. Augmenting SAND with a spherical data model. In *International Conference on Discrete Global Grids*, Santa Barbara, CA, March 2000.

[3] W. G. Aref and H. Samet. Efficient processing of window queries in the pyramid data structure. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 265–272, Nashville, TN, April 1990. Also *Proceedings of the Fifth Brazilian Symposium on Databases*, pages 15–26, Rio de Janeiro, Brazil, April 1990.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD Conference*, pages 322–331, Atlantic City, NJ, June 1990.

[5] B. B. Bederson and J. D. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. *Journal of Visual Languages and Computing*, 7(1):3–31, March 1996.

[6] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the SIGGRAPH'93 Conference*, pages 73–80, Anaheim, CA, August 1993.

[7] T. Brinkhoff and H.-P. Kriegel. The impact of global clustering on spatial database systems. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, J. Bocca, M. Jarke, and C. Zaniolo, eds., pages 168–179, Santiago, Chile, September 1994.

[8] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, June 1979.

[9] C. B. Cranston, F. Brabec, G. R. Hjaltason, D. Nebert, and H. Samet. Adding an interoperable server interface to a spatial database: Implementation experiences with OpenMap™. In *Interoperating Geographic Information Systems — Second International Conference, INTEROP'99*, A. Včkovski, K. Brassel, and H.-J. Schek, eds., pages 115–128, Zurich, Switzerland, March 1999. Also Springer-Verlag Lecture Notes in Computer Science 1580.

[10] M. J. Egenhofer. Why not SQL! *International Journal of Geographical Information Systems*, 6(2):71–85, March-April 1992.

[11] M. J. Egenhofer. Spatial sql: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, February 1994.

[12] M. J. Egenhofer and J. R. Richards. Exploratory access to geographic data based on the map-overlay metaphor. *Journal of Visual Languages and Computing*, 4(2):105–125, June 1993.

[13] S. Gadia and V. Chopra. A relational model and SQL-like query language for spatial databases. In *Advanced Database Systems*, N. R. Adam and B. K. Bhargava, eds., Lecture Notes in Computer Science 759, pages 213–225. Springer-Verlag, Berlin, Germany, 1993.

[14] R. H. Güting. An introduction to spatial database systems. *VLDB Journal*, 3(4):401–444, October 1994.

[15] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference*, pages 47–57, Boston, June 1984.

[16] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In *Proceedings of the ACM SIGMOD Conference*, J. Peckham, ed., pages 171–182, Tucson, AZ, May 1997.

[17] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Advances in Spatial Databases — Fourth International Symposium, SSD'95*, M. J. Egenhofer and J. R. Herring, eds., pages 83–95, Portland, ME, August 1995. Also Springer-Verlag Lecture Notes in Computer Science 951.

[18] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *Proceedings of the ACM SIGMOD Conference*, L. Hass and A. Tiwary, eds., pages 237–248, Seattle, WA, Jun 1998.

[19] 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.

[20] G. Iwerks and H. Samet. The spatial spreadsheet. In *Proceedings of the Third International Conference on Visual Information Systems (VISUAL99)*, D. P. Huijsmans and A. W. M. Smeulders, eds., pages 317–324, Amsterdam, The Netherlands, June 1999.

[21] H. Lieberman. Powers of ten thousand: navigating in large information spaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 15–16, Marina del Rey, CA, November 1994.

[22] 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.

[23] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *Proceedings of the Third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 181–190, Waterloo, Ontario, Canada, April 1984.

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

[25] N. Roussopoulos, C. Faloutsos, and T. Sellis. An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639–650, May 1988.

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

[27] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[28] M. Sarkar and M. H. Brown. Graphical fisheye view of graphs. *Communications of the ACM*, 37(12):73–84, December 1994.

[29] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley, Reading, MA, third edition, 1997.

[30] A. Soffer and H. Samet. Two approaches for integrating symbolic images into a multimedia database systems: a comparative study. *VLDB Journal*, 7(4):253–274, December 1998.

[31] M. Stonebraker. Limitations of spatial simulators for relational DBMSs. Technical report, INFORMIX Software, Inc., 1997. `http://www.informix.com/informix/corpinfo/zines/whitpprs/wpsplsim.pdf`.

[32] C. D. Tomlin. *Geographic information systems and cartographic modelling*. Prentice Hall, Englewood Cliffs, NJ, 1990.

[33] T. Vijlbrief and P. van Oosterom. The GEO++ system: an extensible GIS. In *Proceedings of the Fifth International Symposium on Spatial Data Handling*, pages 40–50, Charleston, SC, August 1992.

[34] W. Wang, J. Yang, and R. Muntz. PK-tree: a spatial index structure for high dimensional point data. In *Proceedings of the Fifth International Conference on Foundations of Data Organization and Algorithms (FODO)*, K. Tanaka and S. Ghandeharizadeh, eds., pages 27–36, Kobe, Japan, November 1998.