Advanced Operations for Maps in Spatial Databases

Mark McKenney and Markus Schneider

University of Florida, Department of Computer & Information Science & Engineering, Gainesville, FL 32611, USA
{mm7, mschneid}@cise.ufl.edu

Summary. Maps are a fundamental spatial concept capable of representing and storing large amounts of information in a visual form. Map operations have been studied and rigorously defined in the literature; however, we identify a new class of *map join* operations which cannot be completed using existing operations. We then consider existing operations involving connectivity concepts, and extend this class of operations by defining new, more complex operations that take advantage of the connectivity properties of maps.

1 Introduction

Spatially-oriented disciplines such as cartography and geography, as well as computer assisted systems like spatial database systems (SDBMS), geographic information systems (GIS) and image database systems rely heavily on the idea of *maps* or *spatial partitions*. A map is a fundamental spatial concept that is capable of representing and storing large amounts of information in a visual form.

Although maps are an intuitive basis for geometric applications such as SDBMS and GIS, these applications do not incorporate maps as a fundamental data type. The fundamental data types are instead geometric data types such as points, lines, and regions, which are compiled together to form map representations. Constructing maps as an amalgam of many more simple types presents the user with a map representation; however, the underlying operations on the map must then be defined based on these more simple types. Thus, maps are not considered "first class citizens" in spatial software systems, but are merely used for visualization purposes. Operations over maps represented as collections of points, lines, and regions are difficult to implement and cannot use the structural information inherent in maps because each individual simple data type has no knowledge of that structure. Furthermore, the regions in maps satisfy certain topological relationships, namely all regions are either disjoint, or share a common boundary. By using regions to represent maps, this constraint cannot be enforced by the data type itself, but must be managed by the system using the maps, such as the SDBMS. By representing maps as a fundamental data type, SDBMS resources do not need to be used to enforce such constraints because

the data type (including the operations defined over that data type) will enforce them implicitly.

Part of the reason that points, lines, and regions are used in spatial systems instead of maps is that manipulating these data types in a database setting is a relatively well understood concept. The basic operations over these objects have been well defined and implemented in various main stream systems. The complex structure of maps, however, causes them to be much more difficult to understand and formalize, as opposed to points, lines, and regions. While much research has been done on map operations, we show scenarios that require new operations that have not previously been considered.

The shortcomings of the currently defined set of map operations rest on three problems. First, the complex structure of maps and the need for operations that take advantage of that structure indicate that a small set of basic operations will not provide enough general functionality to a user. The current set of map operations was defined without considering certain aspects of map structure that a user may wish to take advantage of. For example, there are only a few operations that manipulate maps at the level of their component regions. Secondly, the operations currently defined over maps have been formalized in an ad-hoc fashion. Therefore, in addition to merely introducing new operations, we classify the new operations into general categories that provide some structure to the set of map operations, and will help in further studying map operations. Finally, map operations have largely been defined without considering the embedding of maps into databases. Therefore, some fundamental database concepts have not yet been applied to maps. Such concepts, such as the database join operation, lead to new, powerful map operations.

In this paper, we formalize new operations over maps based on an abstract, point set topological model of maps. While such a model is not suited to discrete representation, it provides an exact mathematical model which can be leveraged to precisely define map operations. Section 2 discusses work related to our research. Section 3 provides motivating query scenarios that highlight the requirements of new map operations. The formal model of spatial partitions and basic operations over them are given in Section 4. Based on this model, new map operations are formalized in Section 5. Finally, in Section 6, we draw some conclusions.

2 Related Work

Attempts in the field of spatial databases have been made to handle maps. Unfortunately, these attempts have been unsatisfactory. In [1], a spatial type *area* is suggested to model constraints on maps. However, the maintenance of these constraints is not supported by the model, but must be enforced by the user. [2] informally proposes a generic data type of partitions called *tessellation* which can be parametrized with an attribute of a yet unspecified type. The concept of *restriction types*, proposed in [3], allows the general type for regions to be restricted to subtypes whose values all satisfy a topological predicate. However, it is unclear how these constraints are controlled by the database.

In this paper, we present operations over maps that are based on the formal model of *spatial partitions* presented in [4]. These operations add to the set of known map operations that have been defined in the literature. For a review of existing map operations, the reader is directed to [5, 6, 7, 8]. While these papers result in a large number of map operations, operations that deal specifically with the structure of maps are generally overlooked.

The study of the traditional data types of point, line, and region and their use in databases [9] is closely related to the concept of maps in databases. For example, the component regions of maps are similar to complex spatial region models and the border structure is similar to complex spatial lines. Furthermore, the regions in maps must satisfy certain topological predicates [10, 11] with each other, namely they can only meet or be disjoint.

3 Motivating Query Scenarios

In this section, we consider map operations from the user's perspective. We assume that a user will have access to a system that provides map access and the ability to perform operations over maps (i.e., a map based GIS system or a map database). A series of scenarios is presented in which the user requires certain information from the maps in the system. We then specify, in English sentences, the specific results required by the user in such scenarios. By examining these specifications in later sections, we discover the need for new map operations.

Throughout this paper, we study map operations as they pertain to specific examples; however, the operations presented are general operations that are relevant to situations other than the ones we show here. In fact, we find that we can define only a few powerful operations that can be generalized to apply to many cases through the use of different operands. We merely present example scenarios to motivate the utility of the operations and to help explain their semantics. For the sake of clarity, we will only introduce three sample maps, shown in Fig. 1, against which queries requiring map operations will be posed. The first map in Fig. 1a depicts counties around a body of water. Each county is labeled with an identifying letter, and the number of flu vaccine units currently in that county. County G contains a bridge over the water which is represented by an extension of G that touches county E. Figure 1b contains regions enclosed by dashed lines which represent areas of high population density. The third map, shown in Fig. 1c shows the areas of high population density superimposed over the map of counties. Operations will be performed over the first two maps, the third is shown for reference. Note that the water shown in the figures is not part of the actual map stored in the database, but is shown merely to provide additional context.

3.1 Map Joining Scenarios

For the remainder of this paper, we will assume that the flu virus has been detected in all of the areas of high population density represented in Fig.1b. Frequently, supplies

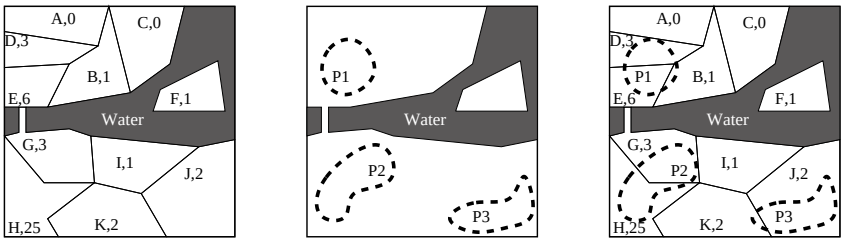

Fig. 1. Figure **a** shows a map of counties. Figure **b** shows areas of high population density. Figure **c** shows **b** superimposed over **a**.

such as flu vaccines are distributed through local governments, such as county governments in the United States. Therefore, state officials need to know which counties contain the infected areas of high population density so that additional vaccinations may be sent there. The officials can obtain this information by combining information from the population density map and the county map to calculate a map containing only counties that overlap a region of high population density (the term 'overlap' indicates the topological predicate *overlap* between two complex regions). This type of query cannot be calculated using existing map operations. Therefore, a new operation is required that allows a user to create a map consisting of complete faces of regions from two source maps that satisfy a given predicate. We call this type of operation a *map join*. The result of such a scenario is shown in Fig. 2a. Only regions from the county map that overlap a region in the population density map remain in the result map. Note is that this particular example represents a one-sided join, i.e., only regions from one of the argument maps are being returned. Two sided joins are also possible, in which regions from both argument maps that satisfy the predicate are shown in the result map.

Query 1. *Return the map consisting of all counties from the county map that overlap a region in the population density map.*

The map join operation is parametrized by a user supplied predicate; thus, different queries can be posed over data by simply changing the predicate used in the join. For example, instead of finding counties that overlap a region of high population density, the official might want to know which counties meet a region of high population density. Such a query would identify counties in which the flu is likely to spread. To express this query, we simple replace the word 'overlap' in Query 1 with the word 'meet' . The resulting map is shown in Fig. 2b.

A second map join scenario considers the connectivity information which is inherently represented in a map's structure. Suppose that the particular strain of the flu that the authorities are tracking is highly contagious. In this case, the authorities need to know all counties that are connected through one or more counties to a county that overlaps an infected region of high population density. Such a map (depicted in Fig. 2c) represents all possible counties in danger of becoming infected.

Query 2. *Return the map consisting of all counties that overlap a region of high population density, and all counties that are connected through one or more counties to county that overlaps a high population density region.*

3.2 Complex Connectivity Scenarios

One of the advantages of using maps as a data type is that topological and connectivity information is inherently stored in the map in a visual form. While some work has been done on identifying and defining operations that utilize connectivity information, the extent to which such information can be used has not been explored. Here we consider scenarios in which connectivity calculations are required beyond those offered by existing map operations.

A more complex extension of the basic notion of connectivity is the idea of connectivity through a specific construct. For example, county H in Fig. 1a contains far more units of flu vaccine than the other counties. Assume that a large number of transport vehicles are stationed in county J. A useful query in this situation is to find out if county C is connected to county J through county H. If the connection exists, then the transport vehicles can drive to county H, pick up vaccines, and deliver them to county C which has no vaccines. There are two possible versions of such a query, one that returns the map of counties that connect C and J through H, and one that simply returns a true or a false value. A query that calculates the latter would be stated:

Query 3. *Are counties C and J connected through county H?*

Instead of transporting vaccines around the map, the authorities determine that if the sum of vaccine units in a county and all of its neighboring counties is 6 or greater, that county has access to enough vaccines within a short distance to effectively combat the spread of the virus. A query to calculate which regions have enough units of vaccine close by uses the notion of a region's neighborhood, i.e., the region and the regions immediately surrounding it. The authorities need to know which regions are not part of a neighborhood containing at least 6 vaccine units. The result of this scenario is shown in Fig. 2d.

Query 4. *Return the map consisting of all regions that are not part of a neighborhood containing at least 6 vaccine units.*

4 The Spatial Partition Model

In this section we briefly review the definitions for the formal model of spatial partitions. Furthermore, we review the basic spatial partition operations that form the basis for nearly all existing spatial partition operations. The reader is directed to the literature for a complete list of known partition operations.

A spatial partition, in two dimensions, is a subdivision of the plane into pairwise disjoint *regions* such that each region is associated with an attribute having simple or

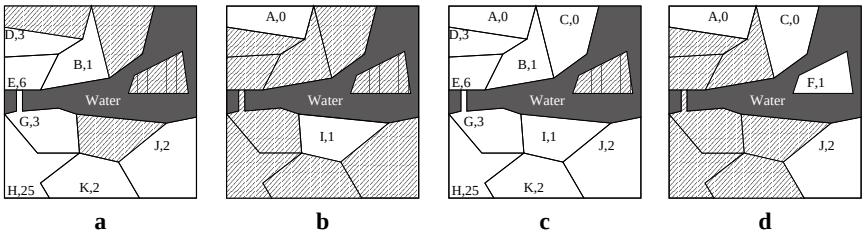

Fig. 2. The resulting partitions from the sample queries. Shaded regions and the water areas are not included in the actual map, but are shown for reference.

complex structure, and these regions are separated from each other by *boundaries*. All points within the spatial partition that have an identical attribute as a particular region are part of that region. Topological relationships are implicitly modeled among the regions in a spatial partition, and can be considered integrity constraints. For instance, neighborhood relationships where different regions share common boundaries are visible in a map. An implication of this property is that, neglecting common boundaries, the regions of a partition are always disjoint; this property causes maps to have a rather simple structure. From this point forward, we use the term "partition" to refer to a spatial partition.

We stated above that each region in a spatial partition is associated with a single attribute. A spatial partition is modeled by mapping Euclidean space to such *attributes* or *labels*. The regions of the partition are then defined as consisting of all points which contain an identical label. Adjacent regions each have unique labels in their interior, but their common boundary is assigned the labels of both regions.

4.1 Formal Spatial Partition Model

We begin by briefly summarizing the mathematical notation used throughout the following sections. The application of a function $f : A \rightarrow B$ to a set of values $S \subseteq A$ is defined as $f(S) := \{f(x)|x \in S\} \subseteq B$. In some cases we know that $f(S)$ returns a singleton set, in which case we write $f[S]$ to to denote the single element, i.e. $f(S) = \{y\} \Longrightarrow f[S] = y$. For doubly nested singleton sets, we use $f[[\cdot]]$ similarly.

The inverse function $f^{-1} : B \rightarrow 2^A$ of f is defined $f^{-1}(y) := \{x \in S| f(x) = y\}$. It is important to note that f^{-1} is a total function and that f^{-1} applied to a set yields a set of sets. We define the range function of a function $f : A \rightarrow B$ as $rng(f) := f(A)$. It is useful to denote the range of a particular partition. Thus, for a set-valued function $f : A \rightarrow 2^B$, we define the notation $s\text{-}rng[f] := \{b \in B|\{b\} \in rng(f)\}$ which gives the values occurring in singleton sets.

It is useful to sometimes denote functions as parameters for operations. In these cases, the *Lambda notation* $\lambda x : S.E(x)$ where E is an expression using x is used. This is simply an abbreviation for the set expression $\{(x, E(x))|x \in S\}$.

Let (X,T) be a topological space with topology $T \subseteq 2^x$, and let $S \subseteq X$ [1]. The *interior* of S, denoted by Int$S$, is defined as the union of all open sets that are contained in S. The *closure* of S, denoted by $\overline{S}$ is defined as the intersection of all closed sets that contain S. The *exterior* of S is given by Ext$S := \text{Int}(X - S)$, and the *boundary* or *frontier* of S is defined as $\text{Fr}S := \overline{S} \cap \overline{X - S}$. An open set is *regular* if $A = \text{Int}\,\overline{A}$. The type of regular open sets is closed under intersection. In this paper, we deal with $\mathbb{R}^2$ topological space.

A *partition* of a set S, in naive set theory, is a complete decomposition of the set S int non-empty, disjoint subsets $\{S_i | i \in I\}$, called blocks:

(i) $\forall i \in I : S_i \neq \emptyset$,
(ii) $\bigcup_{i \in I} S_i = S$, and
(iii) $\forall i, j \in I, i \neq j : S_i \cap S_j \neq \emptyset$.

where I is an index set used to name different blocks. A partition can equivalently be regarded as a total and surjective function $f : S \rightarrow I$. However, a partition cannot be defined simply as a set-theoretic partition of the plane, that is, as a partition of $\mathbb{R}^2$ or as a function $f : \mathbb{R}^2 \rightarrow I$, for two reasons: first, f cannot be assumed to be total in general, and second, f cannot be uniquely defined on the borders betwen adjacent subsets of $\mathbb{R}^2$. Furthermore, from an application point of view, it is desirable to to require blocks (which model regions of a common label) to be regular open sets [13].

In [4], spatial partitions have been defined in several steps. First a *spatial mapping* of type A is a total function $\pi : \mathbb{R}^2 \rightarrow 2^A$. The existence of an undefined element $\perp_A$ is required to represent undefined labels (i.e., the exterior of a partition is a block $b \in \mathbb{R}^2$ with $\pi[p] = \perp_A$ for all $p \in b$). The labels on the borders of regions are modeled using the power set 2^A; a *border* of π is a block that is mapped to a subset of A containing two or more elements, as opposed to a *region* of π which is a block mapped to a singleton set. The *interior* of π is defined as the union of π's regions. The *boundary* of π is defined as the union of π's borders.

Definition 1. Let π be a spatial mapping of type A

(i) $\rho(\pi) := \pi^{-1}(rng(\pi) \cap 2^A)$ (*regions*)
(ii) $\omega(\pi) := \pi^{-1}(rng(\pi) \cap 2^A)$ (*borders*)
(iii) $\iota(\pi) := \bigcup_{r \in \rho(\pi)} r$ (*interior*)
(iv) $\beta(\pi) := \bigcup_{b \in \omega(\pi)} b$ (*boundary*)

A *spatial partition* of type A is then defined as a spatial mapping of type A whose regions are regular open sets and whose borders are labeled with the union of labels of all adjacent regions:

Definition 2. A *spatial partition* of type A is a spatial mapping π of type A with:

(i) $\forall r \in \rho(\pi) : r = \text{Int}\overline{r}$
(ii) $\forall b \in \omega(\pi) : \pi[b] = \{\pi[r] | r \in \rho(\pi) \wedge b \subseteq \overline{r}\}$

[1] In topological space, the following three axioms hold [12]: (i) $U, V \in T \Longrightarrow U \cap V \in T$, (ii) $S \subseteq T \Longrightarrow \bigcup_{U \in S} U \in T$, and (iii) $X \in T, \emptyset \in T$. The elements of T are called *open sets*, their complements in X are called *closed sets*, and the elements of X are called *points*.

4.2 Basic Partition Operations

Three basic spatial partition operations have been defined that can be used to form the formal definitions of almost all other known partition operations: *intersection*, *relabel*, and *refine*. Each of these operations is closed over the set of valid spatial partitions, meaning that if valid partitions are supplied as arguments to these operations, a valid partition will be returned. The intersection of two partitions π and σ, of types A and B respectively, returns a spatial partition of type $A \times B$ such that each interior point p of the resulting partition is mapped to the pair of labels $(\pi[p], \sigma[p])$, and all border points are mapped to the set of labels of all adjacent regions. Formally, the definition of intersection of two partitions π and σ of types A and B can be described in several steps. First, the regions of the resulting partition must be known. This can be calculated by a simple set intersection of all regions in both partitions, since $\cap$ is closed on regular open sets.

$$\rho_{\cap}(\pi, \sigma) := \{r \cap s | r \in \rho(\pi) \wedge s \in \rho(\sigma)\}$$

The union of all these regions gives the interior of the resulting partition: $\iota_{\cap}(\pi, \sigma) := \cup_{r \in \rho_{\cap}(\pi,\sigma)} r$. Next, the spatial mapping restricted just to the interior is calculated by mapping each interior point $p \in I := \iota_{\cap}(\pi, \sigma)$ to the pair of labels given by π and σ:

$$\pi_I := \lambda\, p : I.\{(\pi[p], \sigma[p])\}$$

Finally, the boundary labels are derived from the labels of all adjacent regions. Let $R := \rho_{\cap}(\pi, \sigma)$, $I := \iota_{\cap}(\pi, \sigma)$, and $F := \mathbb{R}^2 - I$. Then we have:

$$\begin{gathered} \mathit{intersection} : [A] \times [B] \rightarrow [A \times B] \\ \mathit{intersection}(\pi, \sigma) := \pi_I \cup \lambda\, p : F.\{\pi_I[[r]] | r \in R \wedge p \in \bar{r}\} \end{gathered}$$

Relabeling a partition π of type A by a function $f : A \rightarrow B$ is defined as $f \circ \pi$, i.e., in the resulting partition of type B each point p is mapped to $f(\pi(p))$ (recall that $\pi(p)$ yields a singleton set, e.g. $\{a\}$, and that f applied to this yields the singleton set $\{f(a)\}$).

$$\begin{gathered} \mathit{relabel} : [A] \times (A \rightarrow B) \rightarrow [B] \\ \mathit{relabel}(\pi, f) := \lambda\, p : \mathbb{R}^2.f(\pi(p)) \end{gathered}$$

The refinement of a partition identifies the connected components of the partition. This is achieved by relabeling the connected components of a partition with consecutive numbers. A connected component of an open set S is a maximum subset $S \subseteq T$ such that any two points of T can be connected by a curve lying completely inside T [12]. Let $\gamma(r) = \{c_1, ...c_{n_r}\}$ denote the set of connected components in a region r. Then, *refine* can be defined in several steps. The regions of the resulting partition are the connected components of all regions of the original partition:

$$p_{\gamma}(\pi) := \bigcup_{r \in \rho(\pi)} \gamma(r)$$

The union of all these regions results in the interior of the resulting partition: $\iota_\gamma(\pi) := \cup_{r \in \rho_\gamma(\pi)} r$. This means that the set of interior and boundary points are not changed by refine.

We can now define the resulting partition on the interior:

$$\pi_I := \{(p, \{\pi[p], i)\}) | r \in \rho(\pi) \wedge \gamma(r) = \{c_1, ..., c_{n_r}\} \wedge i \in \{1, ..., n_r\} \wedge p \in c_i\}$$

Finally, we derive the labels for the boundary from the interior, much like the definition for intersection. Let $R := \rho_\gamma(\pi)$, $I := \iota_\gamma(\pi)$, and $F := \mathbb{R}^2 - I$. Then:

$$\begin{gathered} \mathit{refine} : [A] \to [A \times \mathbb{N}] \\ \mathit{refine}(\pi) := \pi_I \cup \lambda\, p : F.\{\pi_I[[r]] | r \in R \wedge p \in \bar{r}\} \end{gathered}$$

5 Formalization of Novel Operations

The scenarios presented in Section 3 provide specific instances in which new types of operations are required. In this section, we define new, operations which provide the functionality to calculate those queries. Note that while we used the example queries to motivate the need for these operations, we define operations to satisfy the general types of queries that the examples represent.

5.1 Formalization of Map Join Operations

To calculate Query 1, the query must collect entire regions from one map that satisfy a particular topological predicate with a region from the opposite map. In effect, an operation must calculate a new map of entire regions from argument maps by joining two maps according to a specific constraint. We denote the set of operations that perform this function *map join* operations.

We define the map join in two parts. First, we must define a method by which regions in one of the argument partitions that satisfy the join constraint can be collected. We achieve this in the *collect* operation which takes two partitions, π and σ, and a predicate P (which takes two complex regions), and relabels any region from π that does not satisfy the predicate with at least one region from σ to $\perp$.

$$\begin{gathered} \mathit{collect} : [A] \times [B] \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \to [A] \\ \mathit{collect}(\pi, \sigma, P) := \mathit{relabel}(\pi, \lambda x : A.\ \text{if } \exists y \in B : P(\overline{\pi^{-1}(x)}, \overline{\sigma^{-1}(y)}) \text{ then } x \text{ else } \perp_A) \end{gathered}$$

The join operation can then be defined using the operations *intersection* and *collect*. Since a single region may consist of multiple faces within the partition, we use the refine operation to break such regions so that each face is considered a single region. Because refine extends each region label by appending an integer, we will also use a relabel operation after each collect operation that simply removes the appended integer from each region. We denote this operation *truncate* and omit a formal definition due to space considerations. Given two partitions π and σ and a predicate P, we define the *left-join* and *right-join* operations which create a result partition consisting

of regions from the first and second argument partitions, respectively (i.e., they are one-sided joins):

$$\begin{aligned}
&\textit{left-join} : [A] \times [B] \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \longrightarrow [A] \\
&\textit{left-join}(\pi, \sigma, P) := \textit{truncate}(\textit{collect}(\textit{refine}(\pi), \textit{refine}(\sigma), P)) \\
&\textit{right-join} : [A] \times [B] \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \longrightarrow [B] \\
&\textit{right-join}(\pi, \sigma, P) := \textit{truncate}(\textit{collect}(\textit{refine}(\sigma), \textit{refine}(\pi), P))
\end{aligned}$$

The two-sided join, which we denote simply as *join*, returns the partition consisting of all regions from both argument partitions π and σ that satisfy a given predicate P.

$$\begin{aligned}
&\textit{join} : [A] \times [B] \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \longrightarrow [A \times B] \\
&\textit{join}(\pi, \sigma, P) := \textit{intersection}(\textit{left-join}(\pi, \sigma, P), \textit{right-join}(\pi, \sigma, P))
\end{aligned}$$

These three map join operations are general enough to calculate any join that can be expressed based on a predicate that is satisfied between regions from opposing partitions. However, Query 2 introduces the idea of incorporating connectivity with a map join operation. The result partition for this query must contain the regions that satisfy a left-join operation using the overlap predicate, and regions from the left partition that are connected (through one or more partitions) to the result of the left-join. For example, to compute Fig. 2c, Query 2 must first compute the partition in Fig. 2a using a left-join with the overlap predicate, then add more regions from the county map to the result if they satisfy a second predicate with another region in the county map that is part of the left-join, namely if they are connected to a region in the result of the join. We denote this new operation the *extended-join* between two partitions. To define this new operation, we must use the *difference* [4] operation, which takes two partitions and returns the first partition minus any area that is overlapped by the second partition. We begin by specifying the *left-extended-join* (*right-extended-join*). Given two partitions, π and σ, a predicate P for the join, and a predicate Q for the extended portion of the join, we first calculate the left-join (right-join) based on predicate P. We then include, using the operations intersection and collect, any regions in the difference of π (σ) and the result of the left-join (right-join) that satisfy the predicate Q with a region in the left-join (right-join).

$$\begin{aligned}
&\textit{extended-left-join} : [A] \times [B] \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \longrightarrow [A] \\
&\textit{extended-left-join}(\pi, \sigma, P, Q) := \textit{intersection}(\textit{collect}(\textit{difference}(\pi, \\
&\quad \textit{left-join}(\pi, \sigma, P)), \textit{left-join}(\pi, \sigma, P), Q), \textit{left-join}(\pi, \sigma, P)) \\
&\textit{extended-right-join} : [A] \times [B] \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \longrightarrow [B] \\
&\textit{extended-right-join}(\pi, \sigma, P, Q) := \textit{intersection}(\textit{collect}(\textit{difference}(\sigma, \\
&\quad \textit{right-join}(\pi, \sigma, P)), \textit{right-join}(\pi, \sigma, P), Q), \textit{right-join}(\pi, \sigma, P))
\end{aligned}$$

The two-sided extended join can then be defined as the intersection of the left and right extended joins. Because the second predicate used by the left and right

extended joins is evaluated between regions of the first and second argument partition, respectively, we pass two additional predicates to the two-sided extended join, one which is used by the first argument partition, and the other which is used by the second argument partition.

$$\begin{aligned}
&\textit{extended-join} : [A] \times [B] \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \\
&\quad \times (2^{\mathbb{R}^2} \times 2^{\mathbb{R}^2} \to \mathbb{B}) \longrightarrow [A \times B] \\
&\textit{extended-join}(\pi, \sigma, P, Q, R) := \textit{intersection}(\textit{extended-left-join}(\pi, \sigma, P, Q), \\
&\quad \textit{extended-right-join}(\pi, \sigma, P, R))
\end{aligned}$$

Query 2 can be computed using an extended left join operation. However, to complete the query, a predicate is required that takes two regions in a partition and returns a value of true if those regions are connected, and returns a value of false otherwise. This notion of connectivity has been studied in previous work; however, we will provide a slightly more general definition of connectivity. We define the *connected* predicate that takes two regions r and s each belonging to a separate argument partition. This predicate returns a value of true if there exists a chain of regions from r to s such that each region in the chain is not disjoint with its neighbors in the chain (i.e., the regions share at least one common point). Furthermore, each region in the chain can be from either argument partition. For example, if the two argument partitions were the county map and the population density map, then the chain J, P3, K, H, P2 connects regions J and P2. Given two regions r and s from partitions π and σ respectively, we define the connected predicate as follows:

$$\begin{aligned}
&\textit{connected} : [A] \times A \times [B] \times B \longrightarrow \mathbb{B} \\
&\textit{connected}(\pi, r, \sigma, s) := \\
&\begin{cases}
\text{true} & \text{if } \exists t_1 \ldots t_n | t_1 = r \wedge t_n = s \wedge \forall 1 \leq i < n : \\
& ((t_i, t_{i+1} \in A \wedge \pi^{-1}(t_i), \pi^{-1}(t_{i+1}) \in \rho(\pi) \wedge \neg \textit{disjoint}(\overline{\pi^{-1}(t_i)}, \overline{\pi^{-1}(t_{i+1})})) \\
& \vee (t_i, t_{i+1} \in B \wedge \sigma^{-1}(t_i), \sigma^{-1}(t_{i+1}) \in \rho(\sigma) \wedge \neg \textit{disjoint}(\overline{\sigma^{-1}(t_i)}, \overline{\sigma^{-1}(t_{i+1})})) \\
& \vee (t_i \in A \wedge t_{i+1} \in B \wedge \pi^{-1}(t_i) \in \rho(\pi) \wedge \sigma^{-1}(t_{i+1}) \in \rho(\sigma) \\
& \wedge \neg \textit{disjoint}(\overline{\pi^{-1}(t_i)}, \overline{\sigma^{-1}(t_{i+1})})) \vee (t_i \in B \wedge t_{i+1} \in A \wedge \sigma^{-1}(t_i) \in \rho(\sigma) \wedge \\
& \pi^{-1}(t_{i+1}) \in \rho(\pi) \wedge \neg \textit{disjoint}(\overline{\sigma^{-1}(t_i)}, \overline{\pi^{-1}(t_{i+1})}))) \\
\text{false} & \text{otherwise}
\end{cases}
\end{aligned}$$

The purpose of Query 2 is to find all counties in the county map that the flu is likely to spread to in the future based on the connectivity of a county to a county which already has flu infections present in it. However, no limit is set on the number of counties in a connected chain. Instead, the user may wish to calculate an extended join containing counties that are connected to an infected county by only one or two counties. This query allows the user to see all infected counties, and all counties which are likely to experience flu infections in the near future. To define such a predicate, we introduce the concept of degree of connectivity between two regions.

The *degree of connectivity* between two regions indicates the minimum number of regions, not counting the source and destination regions, that are needed in a

particular partition to connect a source region and a destination region. If two regions with labels r and s in a partition π are not connected, then their connectivity degree is -1. If they are connected directly (meaning they are not disjoint, or they share at least one common point), then their connectivity degree is 0. If a single region connects them, then their connectivity degree is 1, etc. Note that some regions may be connected by more than one path of regions, in which case the connectivity degree is the degree of the minimal path. For example, counties H and K in the county map have a connectivity degree of zero even though they are connected by paths *H,K* and *H,G,I,J,K*.

$$
\begin{aligned}
&\textit{degree-connected} : [A] \times A \times A \longrightarrow \mathbb{N} \\
&\textit{degree-connected}(\pi, r, s) := \\
&\quad \begin{cases}
-1 & \text{if } \neg \textit{connected}(\pi, r, \pi, s) \\
0 & \text{if } \textit{connected}(\pi, r, \pi, s) \wedge \neg \textit{disjoint}(\overline{\pi^{-1}(r)}, \overline{\pi^{-1}(s)}) \\
n & \text{if } \exists t_1, \ldots, t_n \in A | \pi^{-1}(t_1), \ldots, \pi^{-1}(t_n) \in \rho(\pi) \wedge \textit{connected}(\pi, r, \pi, s) \\
 & \wedge \neg \textit{disjoint}(\overline{\pi^{-1}(r)}, \overline{\pi^{-1}(t_1)}) \wedge \ldots \wedge \neg \textit{disjoint}(\overline{\pi^{-1}(t_n)}, \overline{\pi^{-1}(s)}) \\
 & \text{where } n \text{ is minimal}
\end{cases}
\end{aligned}
$$

5.2 Formalization of Complex Connectivity Operations

The notions of connectivity defined in the previous section build upon the idea of determining if two regions in a map are connected. In this section, define new, more advanced connectivity operations.

Query 3 asks if two regions are connected through a specific region in the map. In order to determine whether two regions are connected through a specified region, we define the operation *degree-connected-through*, which takes a partition, a source and destination region label, and a region label which identifies a region that the connection must pass through. The degree of connectivity returned is the minimum number of regions, including the region required to be part of the path, that are needed to connect the source and destination regions, or -1 if such a connection does not exist:

$$
\begin{aligned}
&\textit{degree-connected-through} : [A] \times A \times A \times A \longrightarrow \mathbb{N} \\
&\textit{degree-connected-through}(\pi, r, s, q) := \\
&\quad \begin{cases}
-1 & \text{if } \neg \textit{connected}(\pi, r, \pi, q) \\
 & \quad \vee \neg \textit{connected}(\pi, q, \pi, s) \\
0 & \text{if } \textit{degree-connected}(\pi, r, s) = 0 \\
 & \quad \wedge (r = q \vee s = q) \\
\textit{degree-connected}(\pi, r, q) & \\
\quad + \textit{degree-connected}(\pi, s, q) & \text{if } (r = q \vee s = q) \\
\textit{degree-connected}(\pi, r, q) & \\
\quad + \textit{degree-connected}(\pi, s, q) + 1 & \text{otherwise}
\end{cases}
\end{aligned}
$$

It is sometimes useful to calculate the partition containing the connection of two regions. We use the *connect* operation to provide a partition containing a source and destination region, and the minimum number of regions that connect them. This operation has been previously defined in [5], so we omit a detailed description. The

connect-through operation calculates the partition consisting of a source and destination region, and the minimum number of regions that connect them, including a specified region. This operation can be reduced to a relabeling problem if the set of region labels contained in the resulting partition is known (i.e., the set of labels that correspond to the minimal chain of regions connecting a source and destination region). Using the degree of connectivity, we first define the operation *connected-through-set* which calculate this set of region labels. The connect-through operation then relabels any region in the argument partition whose label is not in that set to the empty label:

$$
\begin{aligned}
&\textit{connected-through-set} : [A] \times A \times A \times A \longrightarrow A \\
&\textit{connected-through-set}(\pi, r, s, q) := \{t_1, \ldots, t_n | \\
&\quad n = \textit{degree-connected-through}(\pi, r, s, q) + 2 \wedge t_1 = r \wedge t_n = s \wedge \\
&\quad (\exists 1 \leq i \leq n : t_i = q) \wedge \forall 1 \leq k \leq n : \pi^{-1}(t_k) \in \rho(\pi) \\
&\quad \wedge \forall 1 \leq j < n : \neg \textit{disjoint}(\overline{\pi^{-1}(t_j)}, \overline{\pi^{-1}(t_{j+1})})\} \\
&\textit{connect-through} : [A] \times A \times A \times A \longrightarrow [A] \\
&\textit{connect-through}(\pi, w, x, y) := \\
&\quad \textit{relabel}(\pi, \lambda z : A.\ \text{if } z \in \textit{connected-through-set}(\pi, w, x, y) \text{ then } z \text{ else } \perp_A)
\end{aligned}
$$

A final connectivity operation suggested in Query 4 is the neighborhood operation. In this query, the user is interested in the number of vaccine units available in the immediate neighborhood of a region (which includes that region and all regions adjacent to that region). In general, a neighborhood can be specified to include any regions within a specified degree of connectivity to a source region. For instance, the 1-neighborhood of a region consists of the source region and all regions adjacent to it. The 2-neighborhood consists of the source region and all regions with a connectivity degree of 0 or 1 to the source region. We define the neighborhood operation to take a partition π, a region label r, and an integer indicating the size of the neighborhood desired. The neighborhood is then computed by assigning the labels of all regions that are not in the neighborhood of r to $\perp$:

$$
\begin{aligned}
&\textit{neighborhood} : [A] \times A \times \mathbb{N} \longrightarrow [A] \\
&\textit{neighborhood}(\pi, r, n) := \\
&\quad \textit{relabel}(\pi, \lambda x : A.\ \text{if } \textit{degree-connected}(\pi, r, x) < n \text{ then } x \text{ else } \perp_A)
\end{aligned}
$$

6 Conclusion

In this paper we have defined several new operations on maps. By examining maps from the user's perspective in a spatial database context, we have identified the new class of map join operations, and demonstrated the power of those operations with example scenarios. We have further defined new operations that deal with complex connectivity concepts. These operations, coupled with existing map operations, provide a powerful operational foundation to further explore the use of maps in spatial databases. This research was conducted as part of an effort to integrate maps as fundamental data type into database systems. Future work includes implementing a

model of spatial partitions in a database and implementing the operations over them. Another aspect of this research is the integration of map operations into a query language such as SQL. In addition to operations, we plan to explore predicates over maps in a future paper.

References

1. Güting, R.H.: Geo-relational algebra: A model and query language for geometric database systems. In: Int. Conf. on Extending Database Technology (EDBT). (1988) 506–527
2. Huang, Z., Svensson, P., Hauska, H.: Solving spatial analysis problems with geosal, a spatial query language. In: Proceedings of the 6th Int. Working Conf. on Scientific and Statistical Database Management, Institut f. Wissenschaftliches Rechnen Eidgenoessische Technische Hochschule Zürich (1992) 1–17
3. Güting, R.H., Schneider, M.: Realm-Based Spatial Data Types: The ROSE Algebra. VLDB Journal **4** (1995) 100–143
4. Erwig, M., Schneider, M.: Partition and conquer. In: 3rd Int. Conf. on Spatial Information Theory (COSIT). LNCS 1329, Springer-Verlag (1997) 389–408
5. Erwig, M., Schneider, M.: Formalization of advanced map operations. In: 9th Int. Symp. on Spatial Data Handling. (2000) 8a.3–17
6. Scholl, M., Voisard, A.: Thematic map modeling. In: SSD 1990: Proceedings of the first symposium on Design and implementation of large spatial databases, New York, NY, USA, Springer-Verlag New York, Inc. (1990) 167–190
7. Chan, E.P.F., Zhu, R.: Ql/g: A query language for geometric databases. In: Ist Int. Conf. on GIS in Urban and Environmental Planning. (1996) 271–286
8. Frank, A.U.: Overlay processing in spatial information systems. In: 8th Int. Symp on Computer-Assisted Cartography, AUTOCARTO. (1987) 16–31
9. Schneider, M.: Spatial Data Types for Database Systems - Finite Resolution Geometry for Geographic Information Systems. Volume LNCS 1288. Springer-Verlag (1997)
10. Egenhofer, M.J., Herring, J.: Categorizing binary topological relations between regions, lines, and points in geographic databases. Technical report, National Center for Geographic Information and Analysis, University of California, Santa Barbara (1990)
11. Schneider, M., Behr, T.: Topological relationships between complex spatial objects. ACM Transactions on Database Systems (2006) Accepted for Publication.
12. Dugundi, J.: Topology. Allyn and Bacon (1966)
13. Tilove, R.B.: Set membership classification: A unified approach to geometric intersection problems. IEEE Trans. on Computers **C-29** (1980) 874–883