Formalization of Advanced Map Operations

Martin Erwig & Markus Schneider
FernUniversität Hagen, Praktische Informatik IV
58084 Hagen, Germany
{*erwig, markus.schneider*}*@fernuni-hagen.de*

1 INTRODUCTION

Numerous applications in spatially-oriented disciplines like geography, cartography, and related areas, as well as in computer-assisted systems like geographical information systems and spatial database systems witness the importance of *maps* or *spatial partitions*. A map is a fundamental and well-known metaphor and a widely recognized geometric and topological structure that is capable of carrying a large amount of information.

In a former paper (Erwig *et al.*, 1997) we have layed the foundation for a formal treatment of spatial partitions. This formal framework rests on three basic and powerful partition operations to which all application-specific operations known in the literature can be reduced. In this paper, we identify advanced operations on maps and extend the formal framework accordingly.

1.1 Maps Revisited

A spatial partition is a subdivision of the plane into pairwise disjoint *regions* where regions are separated from each other by *boundaries* and where each region is associated with an attribute having simple or even complex structure. That is, a region with an attribute incorporates all points of a spatial partition having this attribute. A spatial partition implicitly models topological relationships between the participating regions which can be regarded as integrity constraints. First, it expresses neighborhood relationships where different regions may have common boundaries. This property is immediately visible on a map. A second related aspect is that different regions of a partition are always disjoint (if we neglect common boundaries) so that a visual representation of a partition has a very simple structure and is easy to grasp.

The basic idea for modeling a spatial partition is to map the Euclidean space to some *label* or *attribute type*, that is, regions of a partition are assigned single labels. Adjacent regions have different labels in their interior, and a boundary is assigned the pair of labels of both adjacent regions.

A number of application-specific operations has been defined on maps. The most important operation is *overlay* which allows to arrange two partitions with different attribute categories on top of each other and to combine them through geometric intersection into a new partition of disjoint and adjacent regions. Another operation is *reclassify* which retains the geometric structure of the spatial partition and transforms all or some partition attributes to new or modified attributes. The operation *fusion* is a kind of grouping operation with subsequent geometric union. It merges neighbored regions of a partition with respect to partially identical attributes. The geometric union of all regions of a partition is formed by the operation *cover*; it yields a result partition consisting of a single region. With the operation *clipping* we can compute the intersection of a partition and a given rectangular window. The *difference* operation takes two spatial partitions defined over the same attribute domain and computes the geometric difference of their point sets. All the regions of the first partition are maintained in the result partition except for those parts that have the same attributes in both partitions. The task of the operation *superimpose* is to lay the regions of a partition over another partition and to cover and erase parts of the other partition. Finally, the operation *window* allows to retrieve those complete regions of a spatial partition whose intersection with a given window is not empty.

In (Erwig *et al.*, 1997) we have shown that all these application operations (and even generalizations of them) can be reduced to three fundamental operations *intersection*, *relabel*, and *refine*. Intersecting two spatial partitions means to compute the geometric intersection of all regions and to produce a new spatial partition; each resulting region is labeled with the pair of labels of the original two intersecting regions, and the values on the boundaries are derived from these. Relabeling a spatial partition has the effect of changing the labels of its regions. This can happen by simply renaming the label of each region; or, in particular, distinct labels of two or more regions are mapped to the same new label. If some of these regions are adjacent in the partition, the border between them disappears, and they are fused in the result partition. Refining a partition means to look with a finer granularity on its regions and to reveal and to enumerate the internal component structure of regions.

1.2 New Applications

Despite all these operations coping with a large number of map applications, there are queries that cannot be answered by them and that require a new class of advanced map operations. Assume that we are given a country map with the population number for each country. Then we can ask, for instance, for the total population of all countries. This query needs a traversal over all countries and a simultaneous summation of all population numbers. We provide the operation *map aggregation* for calculating such kinds of statistics over a map. Or we are interested in labeling each country with its region's area. We cannot perform this operation by a simple relabeling since it is applied only to labels of single points and not to whole regions. For this purpose we offer an operation called *region-based map annotation*.

In addition, the advanced operations comprise *map selection* for extracting regions with selected attributes into a new map, *map layering* spreading map information to different layers by distributing and grouping attributes, *map joining* for combining maps with identical geometries but different attributes, *map lookup* searching for map information by attribute patterns, *map annotation* for adding information to the regions of a map that is given by tables (functions), and *path extraction* for finding a path between two regions of a map.

With two exceptions (mentioned later) our model does not leave the context of maps. Hence, the only support we need at a user interface are partitions and a facility to enter arguments. As a consequence, we will obtain simple and user-friendly interfaces.

Section 2 discusses related work. Sections 3 to 6 introduce the advanced map operations in more detail and give application examples for them. Section 7 briefly reviews the formal model of maps and map operations as described in (Erwig *et al.*, 1997). Section 8 formalizes definitions of the new map operations. Finally, Section 9 draws some conclusions.

2 RELATED WORK

Maps have been identified as a central *spatial concept* (Frank, 1990) to organize our perception and understanding of space. They correspond to the cognitive experience and knowledge humans have of areal phenomena in the real world.

Frequently, maps arise from classifying space according to some attribute (like rural areas according to their agricultural use). They are then called *thematic maps* or *categorical coverages* (Frank *et al.*, 1997; Volta *et al.*, 1993). The operations in this context focus only on partitions of attribute values alone; spatial operations on maps including geometric intersections are completely ignored. In categorical coverages, themes and attributes are fixed. This means that dynamic extensions or combinations of different partitions are not possible.

In geographically-oriented applications and systems maps are regarded as the primary tool for spatial analysis tasks (Berry, 1987; Frank, 1987; Frank *et al.*, 1997; Huang *et al.*, 1992; Nagy *et al.*, 1979; Tomlin, 1990; Valenzuela, 1991; Volta *et al.*, 1993). These tasks are solved on the basis of the map operations summarized in the Introduction. Application-oriented expositions of these operations can be found in (Berry, 1987; Dangermond, 1990; Frank, 1987; Güting, 1988; Güting *et al.*, 1995; Huang *et al.*, 1992; Kriegel *et al.*, 1991; Schneider, 1997; Scholl *et al.*, 1989; Tomlin, 1990; Valenzuela, 1991) for *overlay*, in (Berry, 1987; Dangermond, 1990; Huang *et al.*, 1992) for *reclassify*, in (Chan *et al.*, 1996; Güting *et al.*, 1995; Huang *et al.*, 1992; Kriegel *et al.*, 1991; Schneider, 1997; Scholl *et al.*, 1989) for *fusion*, in (Scholl *et al.*, 1989) for *cover*, in (Scholl *et al.*, 1989) for *clipping*, in (Huang *et al.*, 1992) for *difference*, in (Chan *et al.*, 1996; Scholl *et al.*, 1989) for *superimposition*, and in (Scholl *et al.*, 1989) for *window*.

At a first glance, it seems that several advanced operations introduced in this paper can be simply realized with the relational model. This is, indeed, true for thematic data but in no case valid for geometric attributes. The first problem of the relational model is to store geometry in relations. This has led to extended relational data models and to the introduction of spatial data types (Schneider, 1997) as attribute types in relation schemes. A data type for regions is an example. The second problem is that even this approach has not solved the issue how to model the integrity constraints underlying partitions, namely the disjointedness and adjacency of the regions of a partition. These topological constraints cannot be maintained by the relational model so that it is unsafe regarding this aspect. A few unsatisfactory proposals have been made. In (Güting, 1988) a spatial data type *area* is suggested to model constraints on partitions. Within the framework of an extended relational data model the set of polygons occurring in a relation as a column of an attribute of type *area* has to fulfill the integrity constraint that all polygons are adjacent or disjoint to each other. Unfortunately, the maintenance of this property is not supported by the data model, rather it is up to the user's responsibility. A generic data type for partitions, called *tessellation*, is informally introduced in (Huang *et al.*, 1992) as a specialized type for sets of polygons; this type can be parametrized with an attribute of a yet unspecified type. In (Güting *et al.*, 1995) so-called *restriction types* have been proposed. This concept allows one to restrict the general type for regions to subtypes whose values all satisfy a specific topological predicate (like *disjoint*) and which nevertheless inherit the properties and operations of the more general type for regions. But it is unclear which DBMS component controls the adherence to this constraint. The third problem is that it is unclear and probably impossible how to implement geometric operations like intersection with the aid of the relational model.

3 EXTRACTING AND COMBINING MAP LAYERS

A *map layer* (or simply a *layer*) is similar to what is sometimes called a *coverage*, a *thematic map*, an *overlay*, or a *layer* in GIS and cartography. It represents a set of data describing the spatial variation of one or more related attributes in a study area. We will call areas of a map associated with the same attribute *regions*.

Usually, geometric information contained in a map is modeled as a sequence of map layers which can afterwards be overlayed into a single map. In this section, we go the other way round. From an application point of view, we will introduce operations for extracting layers from maps by *map selection* and for recombining layers into a single map by a special kind of *map overlay* called *disjoint map composition*.

As an introductory example we assume a map which presents a classification of land use. The classification comprises attributes like “wheat”, “steel industry”, “coal mining”, “barley”, “chemical industry”, “vegetable”, etc. (see Figure 1).

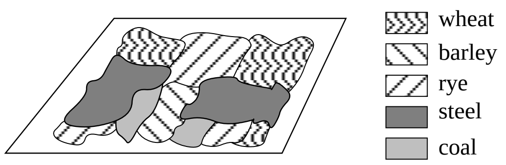


Figure 1: Example of a map

Then we can ask, for instance, for only those areas that are only cultivated with wheat or barley; that is, the use of the remaining areas is not of interest for us. This query requires a map operation called *map selection* which extracts only those regions of the original map which have an attribute out of a collection of pre-specified attributes. The result is a new single map (Figure 2).

Map selection can be generalized to an operation called *map layer generation*. Imagine that we plan to distribute the information of our map from Figure 1 and to show regions of agricultural and industrial use on two separate map layers. Then we group the attributes indicating agricultural use into the more general category “agricultural” and the attributes indicating industrial use into the more general category “industrial”. (In general, we can, of course, use more than two attribute categories.) Afterwards, based on the original map, for each of the two general categories a separate map layer is produced, and each map layer contains all those regions belonging to a general category and labeled according to the original attribute resolution. Figure 3 presents the result of map layer generation based on the map of Figure 1. The general categories are visualized as strings beside the map layers. Map layer generation is, in particular, interesting for spatial analysis tasks solved with the aid of user interfaces where it can be employed by the user to produce and manage new collections of maps.

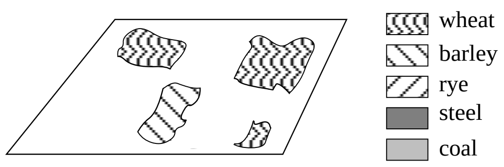


Figure 2: Map after map selection

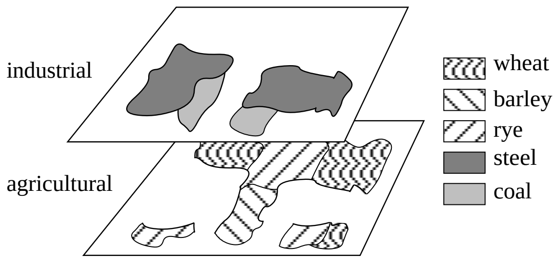


Figure 3: Map after map layer generation

Obviously, due to the described construction process of map layers, the geometries of all map layers are disjoint with respect to their interiors. Only boundaries of regions of different layers may partially coincide. Hence, the original map can be recombined by a *map overlay* operation which does not need to compute any intersections but simply form their geometric union and adopt their corresponding attributes. We call it *disjoint map composition*. From this point of view, disjoint map composition is the inverse operation of map layer generation.

4 JOINING AND INSPECTING MAPS

Spatial analysis frequently produces collections of different maps whose spatial reference system and whose subdivision of space into regions (that is, whose geometry) is the same. These maps essentially differ in the themes they deal with and hence in the attribute distributions. An interesting task is therefore to integrate all attribute informations with respect to the same regions into a single map.

Consider as an example the European countries and assume that a collection of maps gathers statistical data for each country involving country name, population, population density, average income, unemployment rate, spoken language, etc. Figure 4 shows two maps, each map presenting three fictitious countries. The first map is labeled with country names, and the second map is labeled with population density.

Next, we could be interested in an integrated view of all attributes on a single map. The effect is that all attributes are joined; the geometry of the resulting map is the same as the geometry of the original map collection and remains unchanged. We call this operation a *map join*. Figure 5 visualizes the map join of the two maps of Figure 4.

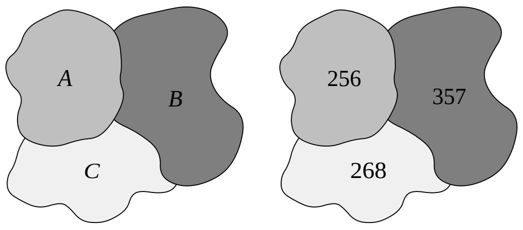


Figure 4: Two example maps with the same geometry but different themes

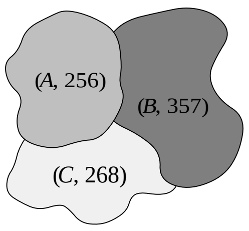


Figure 5: Map after map joining

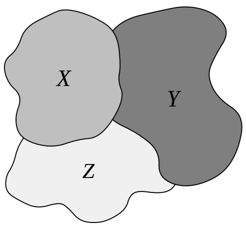
Sometimes, one is interested in the functional relationship between parts of an attribute labeling a region. Concerning our example of Figure 5 we can obtain a function from country name to population density with the instances $A \mapsto 256$, $B \mapsto 357$, and $C \mapsto 268$. We call this operation *functional extraction* since the functional relationship between different attribute parts pertaining to the same region is extracted from a map.

A map usually offers a global view of a collection of regions labeled with some attribute. Often, the user is interested in retrieving only partial information of a map where some kind of "search key" is given as an *attribute pattern* to specify the desired data. Usually, an attribute has a more complex structure and consists of several components. Consider again a map of European countries with the country name, the population, and the spoken language for each country. Assume that we are interested in all countries where German is the spoken language. Then we can search with the attribute pattern (_, _, German) for all these countries. The symbol "_" serves as a wildcard and stands for an arbitrary value of the respective attribute component. Hence, an attribute pattern may have wildcards or concrete values of the corresponding attribute component as entries. In the example, the result is a map containing only those countries speaking German. We call this operation *map lookup*.

5 INFORMATION EXPANSION IN MAPS

Frequently, we are interested in extending the information available on a map. Assume that a map of oil fields with their names is given and that we additionally have a table indicating the owner of each oil field (Figure 6).

To produce a map showing the connection between the name and the owner of an oil field, we have to supplement the name attribute of each oil field appropriately. The result is shown in Figure 7; we combine the map and the table (given as a function) and obtain a new map labeled with the name and the owner for each oil field. This operation is called *map annotation*.



oil field	owner
X	BP
Y	Shell
Z	Exxon

Figure 6: Oil field map and ownership table

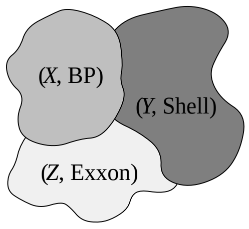


Figure 7: Map after map annotation

So far, annotation in maps has not taken into account the geometry and the geometric properties of the single regions of a map. However, geometry-based annotation leads to very interesting and important queries. For instance, we can ask for the area, the perimeter, or the diameter of each oil field in the map. This necessitates an access to single regions of a map and elementary unary functions computing numerical properties of each region. The oil field map and a function for calculating region areas, for example, are then combined and yield a new map where each region is tagged with the oil field name and its area. We call this operation *region-based map annotation*.

6 MAP AGGREGATION AND CONNECTIVITY

Another important map operation is *map aggregation* which serves for calculating summary statistics over a map or over a distinguished collection of regions of a map. Imagine a map of districts where each district is tagged with its name and its population number (Figure 8).

To compute the total population of all districts, we have to traverse them and to accumulate all population numbers. That is, as input we take the map, a function which adds the current population number to the intermediate sum of the districts already visited, and an initial value for the summation process to start with. In our example, the initial value is 0. The result is a new map (Figure 9) where the aggregation value is attached to each district region of the map.

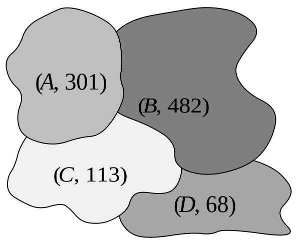


Figure 8: Map showing districts and their population

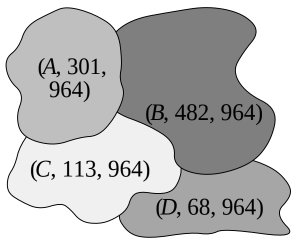


Figure 9: Map showing the total population number attached to all districts

Aggregation values are often used in further map operations. For instance, with the available information we can compute the ratio of each region's population to the total population of all districts and label each district with this result. In our example we obtain the following proportions: $A : 31.22\%, B : 50\%, C : 11.72\%,$ and $D : 7.06\%$.

Besides summation we can, of course, use other numerical aggregation functions. For example, we can compute the minimum average income of a number of countries to identify the poorest country, or the maximum birth or death rate in countries of the Third World.

An example of a non-numerical aggregation function is the insertion of an element into a collection of values. Imagine a country map showing for each country its name, the economic community to which it belongs (like the European Union), and its most important natural resource. The task is to find out all natural resources that are available in an economic community “A”. First, we identify all countries belonging to the economic community “A” by map lookup (see Section 4). Afterwards, we aggregate over this intermediate map and insert all names of natural resources into a set. Hence, the initial value must be the empty set. The result is a new map where all attributes of the original map have been extended by the set resulting from aggregation.

Another interesting issue relates to connectivity properties of maps. This is illustated, for example, by the applications whether it is possible to reach a country K from a country A overland, or whether two friendly nations A and K located on the same continent can visit each other overland without being forced to traverse enemy territory (Figure 10).

These applications ask for a path between A and K. As input we need a country map and two countries A and K for expressing the start and destination of a possible path between them. In any case, the result is a new map (Figure 11).

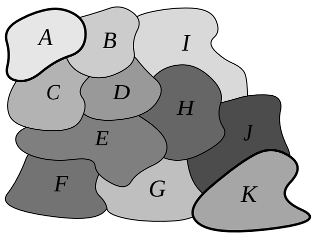


Figure 10: Searching a path on a map

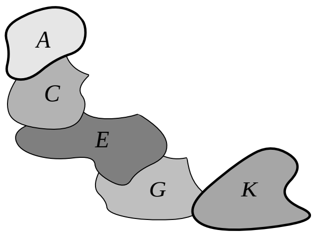


Figure 11: Finding a path on a map

If a path exists, the result is a map with a path of minimal length to indicate the existence of such a path. If a path does not exist, only start and destination (here: the countries A and K) are shown on the result map. We call this operation *path extraction*.

7 SPATIAL PARTITIONS: A FORMAL MODEL OF MAPS

In this section we briefly repeat the definitions of our model for spatial partitions. We provide a precise definition for the type of two-dimensional partitions in Section 7.1 followed by a definition of the basic operations in Section 7.2.

Before we start giving mathematical definitions for partitions, we shortly summarize the used notation. The application of a function $f : A \to B$ to a set of values $S \subseteq A$ is defined as $f(S) := \{f(x) \mid x \in S\} \subseteq B$. If we are sure that $f(S)$ yields a singleton set, we write $f[S]$ to denote this single element (instead of the singleton set), that is, $f(S) = \{y\} \implies f[S] = y$. ($f[S]$ is undefined if $|f(S)| \neq 1$.) Similarly, for doubly-nested singleton sets we use $f[[\cdot]]$ to extract elements, that is, $f(S) = \{\{y\}\} \implies f[[S]] = y$.

Below we frequently have to denote functions used as parameters for operations. For this we employ the *lambda-notation* $\lambda x{:}S.E(x)$ (where E is an expression using x). This is an abbreviation for the set expression $\{(x, E(x)) \mid x \in S\}$ (which actually represents a function).

The *inverse function* $f^{-1} : B \to 2^A$ of f is defined by $f^{-1}(y) := \{x \in S \mid f(x) = y\}$. Note that f^{-1} is a total function and that f^{-1} applied to a set yields a set of sets. The *range* of a function $f : A \to B$ is defined as $rng(f) := f(A)$. Later we frequently have to denote the range of a partition. Therefore, we define for a set-valued function $f : A \to 2^B$ the notation $s\text{-}rng[f] := \{b \in B \mid \{b\} \in rng(f)\}$ giving the values occurring in singleton-sets.

We also introduce a notation for power sets containing sets of constrained size: for $\rhd \in \{>, \geq, =\}$ and $k \in \mathbb{N}$ we define $S^{\rhd k} := \{s \in 2^S \mid |s| \rhd k\}$.

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

A *partition* of a set S can be viewed as a total function $f : S \to I$ into an index set I; f induces an equivalence relationship $\equiv_f$ on S that is defined by $x \equiv_f y \iff f(x) = f(y)$. The equivalence classes $S/{\equiv_f}$ are called *blocks*. The block S_i that corresponds to an index i is given by $S_i := f^{-1}(i)$, and the whole partition $\{S_i \mid i \in I\}$ $(= S/{\equiv_f})$ is also given by $f^{-1}(I)$ if f is surjective.

7.1 The Type of Spatial Partitions

A spatial partition (in the two-dimensional case) is not just defined as a function $f : \mathbb{R}^2 \to I$ for two reasons: first, in most applications f cannot be assumed to be total, and second, f cannot be uniquely defined on borders between adjacent subsets of $\mathbb{R}^2$. Moreover, it is desirable from an application point of view to require blocks (modeling regions of a common label) to be regular open sets (Tilove, 1994).

Therefore, we have defined spatial partitions in several steps (Erwig *et al.*, 1997): first, a *spatial mapping* of type A is a total function $\pi : \mathbb{R}^2 \to 2^A$. We require the existence of an undefined element $\perp_A \in A$, which is used to represent undefined labels, that is, the "exterior" or "outside" of a partition is the block $b \subseteq \mathbb{R}^2$ with $\pi[p] = \perp_A$ for all $p \in b$. The power set range type 2^A is used to model labels on region borders: a *region* of π is a block that is mapped to a singleton set whereas a *border* of π is a block that is mapped to a subset of A containing two or more elements. Then the *interior* of π is defined as the union of π's regions, and 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)	$\rho(\pi) := \pi^{-1}(rng(\pi)^{=1})$	(*regions*)
(ii)	$\omega(\pi) := \pi^{-1}(rng(\pi)^{>1})$	(*borders*)
(iii)	$\iota(\pi) := \bigcup_{r \in \rho(\pi)} r$	(*interior*)
(iv)	$\beta(\pi) := \bigcup_{b \in \omega(\pi)} b$	(*boundary*)

Finally, a *spatial partition* of type A is 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) $\forall r \in \rho(\pi) : r = \mathrm{Int}\,\overline{r}$

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

The use of $\pi[b]$ on the left hand side and $\pi[[r]]$ on the right hand side in (ii) can be made clear as follows: consider a block of a partition π, for example, a region r or a border b. For each point p that is contained

[1]Recall that in a topological space the following three axioms hold (Dugundji, 1966): (i) $U,V \in T \implies U \cap V \in T$, (ii) $S \subseteq T \implies \bigcup_{U \in S} U \in T$, and (iii) $X \in T$, $\varnothing \in T$. The elements of T are called *open sets*, their complements in X are called *closed sets*, and the elements of X are called *points*.

in r or in b, $\pi(p)$ yields as a label a set of values. For $p \in r$, this is a singleton set, say $\{a\}$, and for $p \in b$, this is a set $\{a, b, \ldots\}$ of two or more elements. Now when we apply π to the whole set r (or b), we obtain the set of all labels for all points. By definition these are all equal, so the results of $\pi(r)$ and $\pi(b)$ are $\{\{a\}\}$ and $\{\{a, b, \ldots\}\}$, respectively. Thus, if we want to denote the common label of all points of a block, this is given by $\pi[r] = \{a\}$ or $\pi[b] = \{a, b, \ldots\}$, respectively. Likewise, $\pi[[r]] = a$. Hence, $\pi[b]$ denotes the common label, a set $\{a, b, \ldots\}$, of border block b, and $\pi[[r]]$ gives the label of each touching region.

We denote with $[A]$ the type of all spatial partitions with label type A.

7.2 Operations on Partitions

We have defined three basic operations on spatial partitions: *intersection*, *relabel*, and *refine*. The intersection of two partitions π_1 and π_2 of types A and B, respectively, is again a spatial partition (of type $A \times B$) where each interior point p is mapped to the pair of values $(\pi_1[p], \pi_2[p])$, and all border points are mapped to the set of labels of all adjacent regions (as required by the second part of the definition of a partition). Formally, we can define the intersection of two partitions $\pi_1 : [A]$ and $\pi_2 : [B]$ in several steps: first, we compute the regions of the resulting partition. This can be done by simple set intersection since regions are, by definition, regular open sets and since $\cap$ is closed on regular open sets:

$$\rho_\cap(\pi_1, \pi_2) := \{r \cap r' \mid r \in \rho(\pi_1) \wedge r' \in \rho(\pi_2)\}$$

Second, the union of all these regions gives the interior of the resulting partition: $\iota_\cap(\pi_1, \pi_2) := \cup_{r \in \rho_\cap(\pi_1, \pi_2)} r$. Now the spatial mapping restricted to the interior can be just obtained by mapping each interior point $p \in I := \iota_\cap(\pi_1, \pi_2)$ to the pair of labels given by π_1 and π_2:

$$\pi_I := \lambda p{:}I.\{(\pi_1[p], \pi_2[p])\}$$

Third, the boundary labels can be derived from the labels of all adjacent regions. Let $R := \rho_\cap(\pi_1, \pi_2)$, $I := \iota_\cap(\pi_1, \pi_2)$, and $F := \mathbb{R}^2 - I$. Then we have:

$$\begin{aligned}&\mathit{intersection} : [A] \times [B] \to [A \times B]\\ &\mathit{intersection}(\pi_1, \pi_2) := \pi_I \cup \lambda p{:}F.\{\pi_I[[r]] \mid r \in R \wedge p \in \overline{r}\}\end{aligned}$$

To understand the use of $\pi[[\cdot]]$ in the above definition, recall the remark after Definition 2: since we have to place pairs of labels in the result set and since $\pi(r) = \{\{(a, b)\}\}$, we obtain (a, b) by application of $\pi[[\cdot]]$.

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

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

Finally, the refinement of a partition means the identification of connected components. This is achieved by attaching consecutive numbers to the components. A connected component of an open set S is a maximum subset $T \subseteq S$ such that any two points of T can be connected by a curve lying completely inside T (Dugundji, 1966). Let $\gamma(r) = \{c_1, \ldots, c_{k_r}\}$ denote the set of connected components of a region r. Then, similar to *intersection*, we can define the operation *refine* in several steps.

First, the regions of the resulting partition are the connected components of all regions of the original partition.

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

Again, the union of all these regions gives the interior of the resulting partition: $\iota_\gamma(\pi) := \cup_{r \in \rho_\gamma(\pi)} r$. This means that neither the interior nor the boundary is changed by *refine*.

We can now directly define the resulting partition on the interior, since through the computation of the connected components we automatically obtain a set of numbers that can be used as additional labels. Thus, the refinement of the interior is given by:

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

Finally, we have to derive the labels for the boundary from the interior. (Recall that $\beta(\pi) = \beta(\mathit{refine}(\pi))$.) Now let $R := \rho_\gamma(\pi)$, $I := \iota_\gamma(\pi)$, and $F := \mathbb{R}^2 - I$. Then we have:

$$\begin{aligned} &\mathit{refine} : [A] \to [A \times \mathbb{N}] \\ &\mathit{refine}(\pi) := \pi_I \cup \lambda p{:}F.\{\pi_I[[r]] \mid r \in R \wedge p \in \overline{r}\} \end{aligned}$$

In (Erwig *et al.*, 1997) we have proved that partitions are closed under the three operations *intersection*, *relabel*, and *refine*.

8 ADDITIONAL HIGH LEVEL PARTITION OPERATIONS

The three basic operators presented in the previous section cover a broad range of application-specific operations that have been summarized in the Introduction. In the following subsections we define several additional operations to formalize the advanced and in part novel applications described in Sections 3 to 6.

8.1 Operations for Layering

The *select* operation is used to extract specific parts of a partition. It can be considered, in fact, as a special case of relabeling in which all non-interesting parts of the partition are mapped to undefined and all interesting parts are kept unchanged. The decision, which parts to keep and which to forget, is based on the labels of the partition and is thus realized by a predicate on the label type A. We can therefore define *select* using the operation *relabel*:

$$\begin{aligned} &\mathit{select} : [A] \times (A \to \mathbb{B}) \to [A] \\ &\mathit{select}(\pi, P) := \mathit{relabel}(\pi, \lambda x{:}A.\mathbf{if}\ P(x)\ \mathbf{then}\ x\ \mathbf{else}\ \bot_A) \end{aligned}$$

The "dual" of a partition $\mathit{select}(\pi, P)$ is always given by the expression $\mathit{select}(\pi, \neg P)$. Intuitively, the overlay of $\mathit{select}(\pi, P)$ and $\mathit{select}(\pi, \neg P)$ should always yield the original partition π. To express this relationship formally, we first define an operation *union* that combines two partitions of the same type that are disjoint in the following sense: two partitions $\pi : [A]$ and $\pi' : [A]$ are called *disjoint* iff $\forall p \in \mathbb{R}^2 : \pi[p] = \bot_A \vee \pi'[p] = \bot_A$. We write $\pi \not\cap \pi'$ to express the fact that π and π' are disjoint.

Now *union* is defined to yield the label of either π or π'. If π and π' are not disjoint, *union* is defined to yield $\bot_A$ for all points of their common domain. We can define *union* by relabeling the intersection of π and π' with the following function (for $a, a' \in A$):

$$a \oplus a' := \begin{cases} a' & \text{if } a = \bot_A \\ a & \text{if } a' = \bot_A \\ \bot_A & \text{otherwise} \end{cases}$$

We can now define:

$$\begin{aligned} &\mathit{union} : [A] \times [A] \to [A] \\ &\mathit{union}(\pi, \pi') := \mathit{relabel}(\mathit{intersection}(\pi, \pi'), \lambda(x, y){:}A \times A.x \oplus y) \end{aligned}$$

Thus, we can express the above characterization of *select* as

$$\mathit{union}(\mathit{select}(\pi,P),\mathit{select}(\pi,\neg P)) = \pi$$

which is true because $\mathit{select}(\pi,P) \not\cap \mathit{select}(\pi,\neg P)$.

The operation *layer* is a generalization of *relabel* and constructs layers of partitions according to the relabeling function. In practice, the target type B of the relabeling function $f : A \to B$ will often be used to group sets of A-labels into different classes, and for each class an own partition is computed. More precisely, for each label $b \in B$ let $A_b \subseteq A$ be the set of A-labels mapped by f to b (that is, $A_b = f^{-1}(b)$). Then with $\pi : [A]$, $\mathit{layer}(\pi, f)$ constructs for each $b \in B$ a partition $\pi_b : [A_b]$ which is identical to π on all points mapped to a label in A_b and which yields $\perp_A$ everywhere else. The relationship between each label $b \in B$ and its corresponding layer $[A_b]$ is captured by the fact that *layer* returns a function of type $B \to [A]$.

$$\begin{aligned}&\mathit{layer} : [A] \times (A \to B) \to (B \to [A])\\&\mathit{layer}(\pi, f) := \lambda b{:}B.\mathit{select}(\pi, \lambda x{:}A.\, f(x) = b)\end{aligned}$$

This means that $\mathit{layer}(\pi, f)$ yields for each $b \in B$ an A-partition whose labels are all mapped by f to b. We call functions of type $B \to [A]$ that result from the *layer* operation also *layered partitions* and a partition A_b a *layer*.

We have a similar characterization for layered partitions as for *select*. To express this, we extend the *union* operation from the binary case into an operation *collapse* that is able to aggregate a complete layered partition:

$$\begin{aligned}&\mathit{collapse} : (B \to [A]) \to [A]\\&\mathit{collapse}(L) := \pi_1 \mathit{union} \ldots \mathit{union}\ \pi_n \qquad \text{where } L(B) = \{\pi_1,\ldots,\pi_n\}\end{aligned}$$

With *collapse* we can express the fact that the overlay of a layered partition generated by the operation *layer* from a partition π yields again the partition π:

$$\mathit{collapse}(\mathit{layer}(\pi)) = \pi$$

which is true because the layers generated by *select* are pairwise disjoint.

8.2 Joining and Inspecting Partitions

The *relate* operation allows to access specific region labels of one partition $\pi' : [B]$ by using another partition $\pi : [A]$ to identify the corresponding region. This requires both partitions to have exactly the same regions, that is, $\rho(\pi) = \rho(\pi')$. In that case we can compute a function that gives for each $a \in \mathit{s\text{-}rng}[\pi] \subseteq A$ the value $b \in B$ to which the region is mapped by π'. We can define the *relate* operation by composing π^{-1} with π'.

$$\begin{aligned}&\mathit{relate} : [A] \times [B] \to (A \to B)\\&\mathit{relate}(\pi,\pi') := \lambda x{:}A.\pi'[[\pi^{-1}(\{x\})]]\end{aligned}$$

Since π^{-1} yields a region, that is, a set of points, we have to use $\pi[[\cdot]]$ to extract the single label value. The relationships between π, π', and $\mathit{relate}(\pi,\pi')$ are summarized in Figure 12.

We can well consider partitions as database objects: then the regions serve, in a sense, as object identifiers, and partitions, such as $\pi : [A]$, play the role of attributes (with name π and domain A). In the context of the operation *relate*, we notice that π is used like a key attribute. (Note that by definition, any partition is a key attribute in the sense that the labels identify the regions.)

$$\begin{tikzcd}
 & \mathbb{R}^2 \arrow[dl, shift right, "\pi"'] \arrow[dr, "\pi'"] & \\
A \arrow[ur, shift right, "\pi^{-1}"'] \arrow[rr, "relate(\pi,\pi')"'] & & B
\end{tikzcd}$$

Figure 12: Definition of *relate*

One drawback of the *relate* operation is that the result of the operation, a function of type $A \rightarrow B$, is not a partition. In contrast, all other operations do return partitions (except *layer* which returns a layered partition). Therefore, we favor a different way of accessing partition labels, which is described in the following.

First, we can accumulate different labels for the same regions by simply using the *intersection* operation on corresponding partitions $\pi : [A]$ and $\pi' : [B]$. This results in a partition having again the same regions as π and π', but whose labels are of type $A \times B$. The application of *intersection* to two partitions having the same regions represents an important special case, and we call this operation *join*.

Then the retrieval of specific informations from partitions is achieved by the operation *lookup*, which takes a partition $\pi : [A]$ and a *pattern* of type A, and constructs a (sub-) partition of type A of all those parts whose labels match the pattern. First of all, we have to explain what a pattern is: a pattern is either a value or the wildcard symbol "_". The type of patterns over a (non-product) type A is denoted by $\underline{A}$ and is defined as:

$$\underline{A} := A \cup \{\_\}$$

Thus, for example, 17 and _ are elements of the pattern type $\underline{\mathbb{N}}$. A pattern over a product type is simply a tuple of patterns over the component types, that is,

$$\underline{A_1 \times \ldots \times A_k} := \underline{A_1} \times \ldots \times \underline{A_k}$$

Hence, $(3, true)$ is a pattern of type $\underline{\mathbb{N} \times \mathbb{B}}$, as are $(0, \_)$ or $(\_, \_)$. A pattern specifies the values *lookup* should search for in a partition. The wildcards serve as "don't care"-values that match any value. Formally, matching a pattern ϕ (which can be a wildcard _ or a value y) against a value x, written as $\phi \prec x$, is defined by the following rules:

$$\begin{array}{ll}
\_ \prec x & \\
y \prec x & \iff y = x \\
(\phi_1, \ldots, \phi_k) \prec (x_1, \ldots, x_k) & \iff \phi_1 \prec x_1 \wedge \ldots \wedge \phi_k \prec x_k
\end{array}$$

The first line expresses the fact that a wildcard matches any value, the second line says that values match on equality, and the last line defines that tuples match component-wise.

Now *lookup* can be simply defined by using *select*:

$$\begin{array}{l}
lookup : [A] \times \underline{A} \rightarrow [A] \\
lookup(\pi, \phi) := select(\pi, \lambda x{:}A.\phi \prec x)
\end{array}$$

The result of *lookup* is a partition containing all regions whose labels match the specified pattern.

8.3 Extend Operators

We define two "label extension" operators that address two shortcomings of the *relabel* operation as defined in Section 7.2. These operators realize map annotation.

First, the newly assigned labels are automatically keys of the partition resulting from *relabel*, which leads sometimes to undesired fusion effects. Consider, for example, a user who tries to assign with *relabel* a numeric label to a partition of countries. Now if two countries accidentally get the same label, the regions of the two countries are fused in the partition that is returned by *relabel*, which is generally not what the user expected. A possible solution is to keep the original label together with the newly assigned one. This is actually possible with *relabel* itself, but since this operation will occur quite frequently, we give it an own name. Therefore, we define an operation *extend* as follows:

$$\begin{aligned} &\mathit{extend} : [A] \times (A \to B) \to [A \times B] \\ &\mathit{extend}(\pi, f) := \mathit{relabel}(\pi, \lambda x{:}A.(x, f(x))) \end{aligned}$$

An important property of *extend* is that the regions of the argument partition are not changed. We can express this formally by:

$$\rho(\mathit{extend}(\pi, f)) = \rho(\pi)$$

The second limitation of *relabel* is that it is a *local* operation in the sense that each point is assigned a new label independent from all other points; *relabel* depends only on the old label of each point. In some situations, however, it is important to assign a label to a whole set of points where the label value depends on that set. This is particularly true in cases when numeric labels are to be assigned that depend on regions, such as the area or the diameter of a region. In most cases one is not interested in getting regions of the same label fused. Thus, we define the operation *r-extend* similar to *extend*: the only difference is that the parameter function is not applied to each point's label but to the region the point lies in.

$$\begin{aligned} &\mathit{r\text{-}extend} : [A] \times (2^{\mathbb{R}^2} \to B) \to [A \times B] \\ &\mathit{r\text{-}extend}(\pi, f) := \mathit{relabel}(\pi, \lambda x{:}A.(x, f(\pi^{-1}(\{x\})))) \end{aligned}$$

As for *extend*, the regions of the argument partition are not affected by *r-extend*:

$$\rho(\mathit{r\text{-}extend}(\pi, f)) = \rho(\pi)$$

8.4 Partition Aggregation

In the design of an aggregation operator we have strived for a compromise between a powerful operator that is capable of expressing many interesting map operations and a simple operator that is easy to understand by users. We have arrived at an operator *aggregate* that accumulates the labels of a partition $\pi : [A]$ with a binary function $f : A \times B \to B$ starting with an element $u : B$ (u is called *unit*). In many cases we will have $B = A$, for example, when aggregating numerical labels by min, $+$, and so on. However, there are also cases in which a type $B \neq A$ is needed, for example, when collecting labels in a set; in that case we have: $B = 2^A$. Since the definition of *aggregate* does not have to take the adjacency structure of the map into account, we can use the well-known aggregations of sets (Breazu-Tannen *et al.*, 1991; Erwig *et al.*, 1991; Breazu-Tannen *et al.*, 1992; Fegaras *et al.*, 1995): $agg(f,u,S)$ to denote the aggregation of a finite, non-empty set by a binary function:

$$\begin{aligned} agg(f,u,\varnothing) &= u \\ agg(f,u,\{a\} \cup S) &= f(a, agg(f,u,S)) \end{aligned}$$

Now we can define map aggregation as follows (recall that for $\pi : [A]$ we have $\mathit{s\text{-}rng}[\pi] = \{a \in A \mid \{a\} \in \mathit{rng}(\pi)\}$):

$$\begin{aligned} &\mathit{aggregate} : [A] \times (A \times B \to B) \times B \to [A \times B] \\ &\mathit{aggregate}(\pi, f, u) := \mathit{extend}(\pi, \lambda x{:}A.\mathit{agg}(f, u, \mathit{s\text{-}rng}[\pi])) \end{aligned}$$

Note that we do not just return the accumulated value, but rather extend all regions of the argument partition with the accumulated value. This is in line with our approach to stay within partition types, and it also provides better support for many applications where the accumulated value is, in a further step, related to the regions' values (for example, to show the proportion of an area to the total area).

We could have defined *aggregate* in a much more general way: by taking the dual graph of a partition an aggregation operator can move in an exactly prescribed way through this graph. We have actually defined and investigated several versions of such an operator in a different context (Erwig, 1997). Exploiting adjacency (and implicitly also distance) information of regions clearly makes such aggregation operators more expressive, but they also require some practice to be effectively used.

Instead, we add a further operator to support applications that exploit the topological structure of partitions. We define the operation *connect* that takes a partition $\pi : [A]$ and two labels $x,y : A$ and determines whether the regions labeled x and y are connected, that is, whether the regions are adjacent or whether there is a chain of regions connecting them.

The *dual graph* G_π of a partition $\pi : [A]$ is defined as an undirected, unlabeled graph in which nodes are represented by region labels: $G_\pi = (s\text{-}rng[\pi], E_\pi)$ where $E_\pi = rng(\pi)^{=2}$. Note that an edge is represented by a two-element set of A-labels, and since borders between adjacent regions are uniquely labeled with a set containing both regions' labels, we can simply take the set of all two-element border labels as the set of edges. A *path* in G_π between two nodes x and y is a non-empty sequence of nodes $p = x_1, \ldots, x_n$ with $x = x_1$, $y = x_n$, and $\{x_i, x_{i+1}\} \in E_\pi$ for $i \in \{1, \ldots, n-1\}$. The length of such a path is $\ell(p) := n-1$. We denote by $N_{x,y}(G_\pi)$ the set of nodes contained in a shortest path (with respect to ℓ) between x and y. When no path exists in G_π between x and y, $N_{x,y}(G_\pi)$ is defined to yield $\{x,y\}$.

Now *connect* is defined to return a sub-partition of π that contains all regions of a shortest path between x and y in G_π if it exists and just the regions for x and y otherwise.

$$\begin{aligned} &connect : [A] \times A \times A \to [A] \\ &connect(\pi, x, y) := select(\pi, \lambda z{:}A.z \in N_{x,y}(G_\pi)) \end{aligned}$$

9 CONCLUSIONS

We have identified several new operations on maps. By using a formal model for partitions we provide a theoretical framework that is well-suited for studying maps and operations on them: the definitions become quite simple, where at the same time, the operations are very powerful. In particular, variations and extensions of map operations can be easily defined and studied.

References

Berry, J. K. (1987). Fundamental Operations in Computer-Assisted Map Analysis. *Int. Journal of Geographical Information Systems*, **1**(2), 119–136.

Breazu-Tannen, V., Bunemann, P. and Naqvi, S. (1991). Structural Recursion as a Query Language. *3rd Int. Workshop on Database Programming Languages*. 1–12.

Breazu-Tannen, V., Bunemann, P. and Wong, L. (1992). Naturally Embedded Query Languages. *4th Int. Conf. on Database Theory*. LNCS 646. 140–154.

Chan, E. P. F. and Zhu, R. (1996). QL/G: A Query Language for Geometric Databases. *1st Int. Conf. on GIS in Urban and Environmental Planning*. 271–286.

Dangermond, J. (1990). A Classification of Software Components Commonly Used in Geographic Information Systems. *Introductory Readings in Geographic Information Systems*. Taylor & Francis.

Dugundji, J. (1966). *Topology*. Allyn and Bacon.

Erwig, M. (1997). Functional Programming with Graphs. *2nd ACM Int. Conf. on Functional Programming*. 52–65.

Erwig, M. and Lipeck, U. W. (1991). A Functional DBPL Revealing High Level Optimizations. *3rd Int. Workshop on Database Programming Languages*. 306–321.

Erwig, M. and Schneider, M. (1997). Partition and Conquer. *3rd Int. Conf. on Spatial Information Theory*. LNCS 1329. 389–408.

Fegaras, L. and Maier, D. (1995). Towards an Effective Calculus for Object Query Languages. *ACM SIGMOD Conf. on Management of Data*. 47–58.

Frank, A. U. (1987). Overlay Processing in Spatial Information Systems. *8th Int. Symp. on Computer-Assisted Cartography*. 16–31.

Frank, A. U. (1990). Spatial Concepts, Geometric Data Models and Data Structures. *Computer and Geosciences*.

Frank, A. U., Volta, G. S. and MacGranaghan, M. (1997). Formalization of Families of Categorical Coverages. *Int. Journal of Geographical Information Science*, **11**(3), 215–231.

Güting, R. H. (1988). Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. *Int. Conf. on Extending Database Technology*. LNCS 303. 506–527.

Güting, R. H. and Schneider, M. (1995). Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, **4**(2), 100–143.

Huang, Z., Svensson, P. and Hauska, H. (1992). Solving Spatial Analysis Problems with GeoSAL, a Spatial Query Language. *6th Int. Working Conf. on Scientific and Statistical Database Management*.

Kriegel, H.-P., Brinkhoff, T. and Schneider, R. (1991). The Combination of Spatial Access Methods and Computational Geometry in Geographic Database Systems. *2nd Symp. on Advances in Spatial Databases*. LNCS 525. 5–21.

Nagy, G. and Wagle, S. (1979). Geographic Data Processing. *ACM Computing Surveys*, **11**(2), 139–181.

Schneider, M. (1997). *Spatial Data Types for Database Systems - Finite Resolution Geometry for Geographic Information Systems*. LNCS 1288. Springer-Verlag.

Scholl, M. and Voisard, A. (1989). Thematic Map Modeling. *1st Int. Symp. on Large Spatial Databases*. LNCS 409. 167–190.

Tilove, R. B. (1994). Set Membership Classification: A Unified Approach to Geometric Intersection Problems. *The Computer Journal*, **37**(1), 25–34.

Tomlin, C. D. (1990). *Geographic Information Systems and Cartographic Modeling*. Prentice Hall.

Valenzuela, C. R. (1991). Data Analysis and Modeling. *Remote Sensing and Geographical Information Systems for Resource Management in Developing Countries*. 335–348.

Volta, G. S. and Egenhofer, M. J. (1993). Interaction with Attribute Data Based on Categorical Coverages. *1st Int. Conf. on Spatial Information Theory*. LNCS 716. 215–233.