submitted to Special Issue on: "Integration of Data Mining with Database Technology", Data Mining and Knowledge Discovery, an International Journal, Kluwer Academic Publishers, 1999.

Spatial Data Mining: Database Primitives, Algorithms and Efficient DBMS Support

Martin Ester, Alexander Frommelt, Hans-Peter Kriegel, Jörg Sander

Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany
{ester | kriegel | sander}@informatik.uni-muenchen.de

Abstract: Spatial data mining algorithms heavily depend on the efficient processing of neighborhood relations since the neighbors of many objects have to be investigated in a single run of a typical algorithm. Therefore, providing general concepts for neighborhood relations as well as an efficient implementation of these concepts will allow a tight integration of spatial data mining algorithms with a spatial database management system. This will speed up both, the development and the execution of spatial data mining algorithms. In this paper, we define neighborhood graphs and paths and a small set of database primitives for their manipulation. We show that typical spatial data mining algorithms are well supported by the proposed basic operations. For finding significant spatial patterns, only certain classes of paths “leading away” from a starting object are relevant. We discuss filters allowing only such neighborhood paths which will significantly reduce the search space for spatial data mining algorithms. Furthermore, we introduce neighborhood indices to speed up the processing of our database primitives. We implemented the database primitives on top of a commercial spatial database management system. The effectiveness and efficiency of the proposed approach was evaluated by using an analytical cost model and an extensive experimental study on a geographic database.

1 Introduction

The computerization of many business and government transactions and the advances in scientific data collection tools provide us with a huge and continuously increasing amount of data. This explosive growth of databases has far outpaced the human ability to interpret this data, creating an urgent need for new techniques and tools that support the human in transforming the data into useful information and knowledge. *Knowledge discovery in databases* (*KDD*) has been defined as the non-trivial process of discovering valid, novel, and potentially useful, and ultimately understandable patterns from data [FPS 96]. The process of KDD is interactive and iterative, involving several steps such as the following ones:

- *Selection*: selecting a subset of all attributes and a subset of all data from which the knowledge should be discovered.

- *Data reduction*: using dimensionality reduction or transformation techniques to reduce the effective number of attributes to be considered.
- *Data mining*: the application of appropriate algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data.
- *Evaluation*: interpreting and evaluating the discovered patterns with respect to their usefulness in the given application.

Spatial Database Systems (*SDBS*) (see [Gue 94] for an overview) are database systems for the management of spatial data. To find implicit regularities, rules or patterns hidden in large spatial databases, e.g. for geo-marketing, traffic control or environmental studies, spatial data mining algorithms are very important (see [KHA 96] for an overview of spatial data mining).

Most existing data mining algorithms run on separate and specially prepared files, but integrating them with a *database management system* (*DBMS*) has the following advantages. Redundant storage and potential inconsistencies can be avoided. Furthermore, commercial database systems offer various index structures to support different types of database queries. This functionality can be used without extra implementation effort to speed-up the execution of data mining algorithms (which, in general, have to perform many database queries). Similar to the relational standard language SQL, the use of standard primitives will speed-up the development of new data mining algorithms and will also make them more portable.

In this paper, we introduce a set of database primitives for mining in spatial databases. [AIS 93] follows a similar approach for mining in relational databases. Our database primitives are based on the concept of neighborhood relations since attributes of the neighbors of some object of interest may have an influence on the object itself. The proposed primitives are sufficient to express most of the algorithms for spatial data mining from the literature. We present techniques for efficiently supporting these primitives by a DBMS.

The rest of the paper is organized as follows. Section 2 introduces our database primitives for spatial data mining. In section 3, we review spatial data mining algorithms and demonstrate how they can be expressed by using the proposed primitives. Section 4 presents methods of efficiently supporting our database primitives by existing DBMSs. Section 5 summarizes the contributions and discusses several issues for future research.

2 Database Primitives for Spatial Data Mining

In this section, we introduce a small set of database primitives for spatial data mining (see [EKS 97] for a first sketch). The major difference between mining in relational databases and mining in spatial databases is that attributes of the neighbors of some object of interest may have an influence on the object itself. Therefore, our database primitives are based on the concept of spatial neighborhood relations.

2.1 Neighborhood Relations

The mutual influence between two objects depends on factors such as the topology, the distance or the direction between the objects. For instance, a new industrial plant may pollute its neighborhood depending on the distance and on the major direction of the wind. Figure 1 depicts a map used in the assessment of a possible location for a new industrial plant. The map shows three regions with different degrees of pollution (indicated by the different colors) caused by the planned plant. Furthermore, the influenced objects such as communities and forests are depicted..

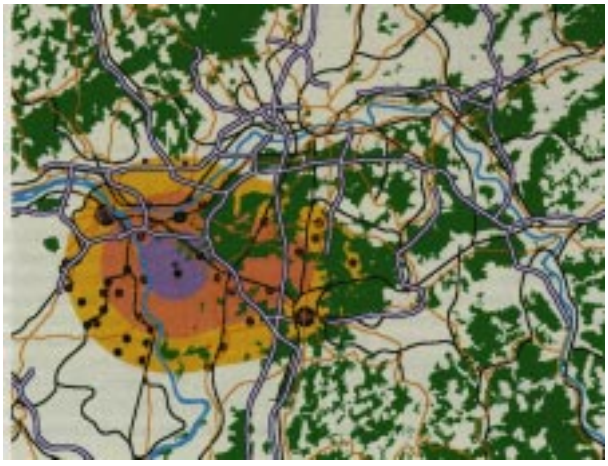


Figure 1. Regions of pollution around a planned industrial plant [BF 91]

In this section, we introduce three basic types of spatial relations: topological, distance and direction relations which are binary relations, i.e. relations between pairs of objects. Spatial objects may be either points or spatially extended objects such as lines, polygons or polyhedrons. Spatially extended objects may be represented by a set of points at its surface, e.g. by the edges of a polygon (vector representation) or by the points contained in the object, e.g. the pixels of an object in a raster image (raster representation). Therefore, we use *sets of points* as a generic representation of spatial

objects. In general, the points $p = (p_1, p_2, \ldots, p_d)$ are elements of a d-dimensional Euclidean vector space called *Points*. In the following, however, we restrict the presentation to the 2-dimensional case, although, all of the introduced notions can easily be applied to higher dimensions d. Spatial objects O are represented by a set of points, i.e. $O \in 2^{Points}$. For a point $p = (p_x, p_y)$, p_x and p_y denote the coordinates of p in the first and the second dimension. $\Delta x(O) := \max\{|o_x - p_x| \mid o, p \in O\}$ is called the *x-extension* of O and $\Delta y(O) := \max\{|o_y - p_y| \mid o, p \in O\}$ the *y-extension* of O.

Topological relations are those relations which are invariant under topological transformations, i.e. they are preserved if both objects are rotated, translated or scaled simultaneously. The formal definitions are based on the boundaries, interiors and complements of the two related objects.

Definition 1: (***topological relations***) The topological relations between two objects A and B are derived from the nine intersections of the interiors, the boundaries and the complements of A and B with each other. The relations are: *A disjoint B, A meets B, A overlaps B, A equals B, A covers B, A covered-by B, A contains B, A inside B.* A formal definintion can be found in [Ege 91].

Distance relations are those relations comparing the distance of two objects with a given constant using one of the arithmetic operators. The distance *dist* between two objects, i.e. sets of points, can then simply be defined by the minimum distance between their points.

Definition 2: (***distance relations***) Let *dist* be a distance function, let σ be one of the arithmetic predicates $<$, $>$ or $=$, let c be a real number and let O_1 and O_2 be spatial objects, i.e. $O_1, O_2 \in 2^{Points}$. Then a distance relation $A\ distance_{\sigma\ c}\ B$ holds iff $dist(O_1, O_2)\ \sigma\ c$.

In the following, we define 2-dimensional direction relations and we will use their geographic names. For dimensions $d > 2$, the number of different direction relations increases but the underlying concepts are still the same.

To define *direction relations* $O_2\ R\ O_1$, we distinguish between the *source object* O_1 and the *destination object* O_2 of the direction relation R. There are several possibilities to define direction relations depending on the number of points they consider in the source and the destination object. We define the direction relation of two spatially extended objects using one representative point $rep(O_1)$ of the source object O_1 and all points of the destination object O_2. The representative point

of a source object may, e.g., be the *center* of the object. This representative point is used as the origin of a virtual coordinate system and its quadrants define the directions.

Definition 3: (***direction relations***) Let *rep*(*A*) be a representative point in the source object *A*.

- *B northeast A* holds, iff $\forall\ b \in B: b_x \geq rep(A)_x \wedge b_y \geq rep(A)_y$

  southeast, *southwest* and *northwest* are defined analogously.

- *B north A* holds, iff $\forall\ b \in B: b_y \geq rep(A)_y$

  south, *west*, *east* are defined analogously.

- *B any_direction A* is defined to be TRUE for all *A*, *B*.

Figure 2 illustrates some of the topological, distance and direction relations using 2D polygons.

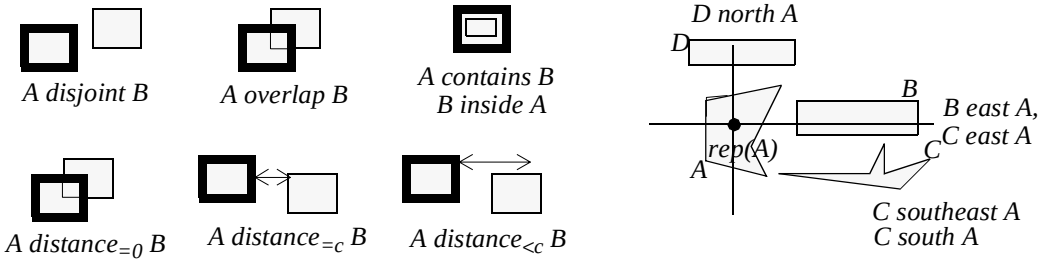


Figure 2. Illustration of the direction relations

Obviously, for each pair of spatial objects at least one of the direction relations holds but the direction relation between two objects may not be unique. Only the special relations *northwest*, *northeast*, *southwest* and *southeast* are mutually exclusive (if we exclude objects with holes, objects with a co-dimension greater than 0, and separations). However, if considering only these special directions there may be pairs of objects for which none of these direction relations hold, e.g. if some points of *B* are northeast of *A* and some points of *B* are northwest of *A*. On the other hand, all the direction relations are partially ordered by a specialization relation (simply given by set inclusion) such that the smallest direction relation for two objects *A* and *B* is uniquely determined. We call this smallest direction relation for two objects *A* and *B* the *exact direction relation* of *A* and *B*.

Topological, distance and direction relations may be combined by the logical operators $\wedge$ (and) as well as $\vee$ (or) to express a *complex neighborhood relation*.

Definition 4: (**complex neighborhood relations**) If r_1 and r_2 are neighborhood relations, then $r_1 \wedge r_2$ and $r_1 \vee r_2$ are also neighborhood relations - called *complex neighborhood relations*.

2.2 Neighborhood Graphs and Their Operations

Based on the neighborhood relations, we introduce the concepts of neighborhood graphs and neighborhood paths and some basic operations for their manipulation.

Definition 5: (***neighborhood graphs and paths***) Let *neighbor* be a neighborhood relation and $DB \subseteq 2^{Points}$ be a database of objects.

a) A *neighborhood graph* $G^{DB}_{neighbor} = (N, E)$ is a graph with the set of nodes N which we identify with the objects $o \in DB$ and the set of edges $E \subseteq N \times N$ where two nodes n_1 and $n_2 \in N$ are connected via some edge of E iff *neighbor*(n_1,n_2) holds. Let n denote the cardinality of N and let e denote the cardinality of E. Then, $f := e / n$ denotes the average number of edges of a node, i.e. f is called the "fan out" of the graph.

b) A *neighborhood path* is a sequence of nodes $[n_1, n_2, \ldots, n_k]$, where *neighbor*(n_i, n_{i+1}) holds for all $n_i \in N, 1 \leq i < k$. The number k of nodes is called the *length* of the neighborhood path.

c) A neighborhood path $[n_1, n_2, \ldots, n_k]$ is *valid* iff $\forall\ i \leq k, j < k:\ i \neq j \Leftrightarrow n_i \neq n_j$.

Lemma 1: The expected number of neighborhood paths of length k starting from a given node is f^{k-1} and the expected number of all neighborhood paths of length k is then $n*f^{k-1}$.

In the following, we will only create *valid* neighborhood paths, i.e. paths containing no cycles. Obviously, even the number of valid neighborhood paths may become very large. For the purpose of KDD, however, we are mostly interested in a certain class of paths, i.e. paths which are "leading *away*" from the starting object in a straightforward sense. We conjecture that a spatial KDD algorithm using a set of paths which are crossing the space in an arbitrary way, leading forward and backwards and contain cycles will not produce useful patterns (if any will be produced at all). Therefore, in addition to our general restriction to valid paths, the operations on neighborhood paths will provide parameters (filters) to further reduce the number of paths actually created.

We will present the signature of the most important operations and a short description of their meaning using the following domains: `Objects`, `NRelations` (neighborhood relations), `Predicates`, `Integer`, `NGraphs` (neighborhood graphs), `NPaths` (neighborhood paths), $2^{\texttt{Objects}}$, $2^{\texttt{NPaths}}$. We do not define an explicit domain of databases. Instead, we use the domain $2^{\texttt{Objects}}$ of all subsets of the set of all objects.

We assume the standard operations from relational algebra such as *selection, union, intersection* and *difference* to be available for sets of objects and for sets of paths. For instance, the operation `selection(db,pred)` returns the set of all elements of a database `db` satisfying the predicate `pred`. We introduce the following basic operations for neighborhood graphs and paths:

`neighbors: NGraphs x Objects x Predicates --> 2`$^{\text{Objects}}$

`extensions: NGraphs x 2`$^{\text{NPaths}}$ `x Integer x Predicates -> 2`$^{\text{NPaths}}$

`paths: 2`$^{\text{Objects}}$ `--> 2`$^{\text{NPaths}}$`;`

`objects: 2`$^{\text{NPaths}}$ `--> 2`$^{\text{Objects}}$

The operation `neighbors(graph,object,pred)` returns the set of all objects connected to `object` via some edge of `graph` satisfying the conditions expressed by the predicate `pred`. The additional selection condition `pred` is used if we want to restrict the investigation explicitly to certain types of neighbors. The definition of the predicate `pred` may use spatial as well as non-spatial attributes of the objects.

The operation `extensions(graph,paths,max,pred)` returns the set of all paths extending one of the elements of `paths` by at most `max` nodes of `graph`. All the extended paths must satisfy the predicate `pred`. Because the number of neighborhood paths may become very large, the operation `extensions` is the most critical operation with respect to efficiency of data mining algorithms. Therefore, the predicate `pred` in the operation `extensions` acts as a filter to restrict the number of paths created using domain knowledge about the relevant paths. Note that the elements of `paths` are not contained in the result implying that an empty result indicates that none of the elements of `paths` could be extended.

The operation `paths(setOfObjects)` creates the set of all paths of length 1 formed by a single element of `setOfObjects`. The operation `objects(setOfPaths)` returns the set of all objects associated with at least one of the nodes of one element of `setOfPaths`.

2.3 Filter Predicates for Neighborhood Paths

Neighborhood graphs will in general contain many paths which are irrelevant if not “misleading” for spatial data mining algorithms. For finding significant spatial patterns, we have to consider only certain classes of paths which are “leading *away*” from the starting object in some straightforward sense. Such spatial patterns are most often the effect of some kind of influence of an object

on other objects in its neighborhood. Furthermore, this influence typically decreases or increases continuously with increasing or decreasing distance. The task of spatial trend analysis, i.e. finding patterns of systematic change of some non-spatial attributes in the neighborhood of certain database objects, can be considered as a typical example. Detecting such trends would be impossible if we do not restrict the pattern space in a way that paths changing direction in arbitrary ways or containing cycles are eliminated.

In the following, we discuss two possible filter predicates, *starlike* and *variable-starlike*. Other filters may be useful depending on the application.

Definition 6: (***filter starlike and filter variable starlike***) Let $p = [n_1,n_2,...,n_k]$ be a neighborhood path and let rel_i be the exact direction for n_i and n_{i+1}, i.e. $n_{i+1}\ rel_i\ n_i$ holds. The predicates *starlike* and *variable-starlike* for paths *p* are defined as follows:

starlike(p) $:\Leftrightarrow (\exists\ j < k: \forall\ i > j: n_{i+1}\ rel_i\ n_i \Leftrightarrow rel_i \subseteq rel_j)$, if $k > 1$; TRUE, if k=1

variable-starlike(p) $:\Leftrightarrow (\exists\ j < k: \forall\ i > j: n_{i+1}\ rel_i\ n_i \Leftrightarrow rel_i \subseteq rel_1)$, if $k > 1$; TRUE, if k=1.

Figure 3. Illustration of two different filter predicates

The filter *starlike* requires that, when extending a path *p*, the exact "final" direction rel_j of *p* cannot be generalized. For instance, a path with "final" direction *northeast* can only be extended by a node of an edge with exact direction *northeast* but not by an edge with exact direction *north.*

The *variable-starlike* filter is less restrictive than the *starlike* filter. It requires only that, when extending a path *p* with a node n_{k+1}, the edge $e = (n_k, n_{k+1})$ has to fulfill at least the exact "initial" direction rel_1 of *p*. Note that $rel_j \subseteq rel_1$ holds if a filter (*starlike* or *variable-starlike*) is used for each extension starting from length 1. For instance, a neighborhood path with "intial" direction *north* can be extended by a node n_{k+1} if *e* satisfies the direction *north* or one of the more special directions *northeast* or *northwest*. Figure 3 illustrates the neighborhood paths for the filters *starlike* and *variable-starlike* when extending the paths from a given starting object.

The exact direction relation *rel* for a source object *A* and a destination object *B* is not independent from their sizes. If *B* is smaller than *A* then *rel* is likely to be a special relation, if *B* is larger

than *A* then *rel* typically is a general direction relation. In the following, we analyze this dependency considering the special but important case of *A* and *B* having the same size. Let *A* be a source object and *B* be a destination object satisfying *A intersect B* and *B south A.* Then, there are three groups of such objects *B* as depicted in figure 4.

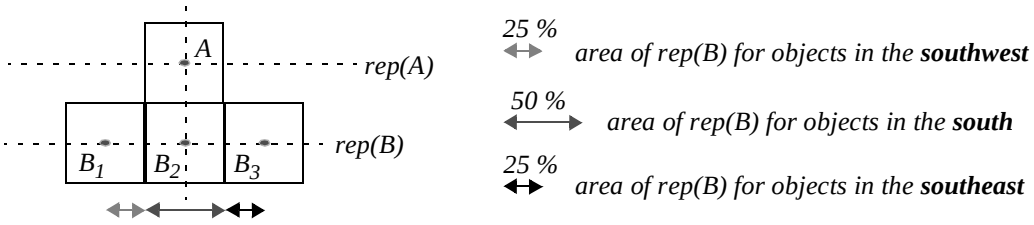


Figure 4. Objects *B* with "*B* south of *A* and *B* intersects *A*"

All objects of the middle group B_2 fulfill the exact direction relation B_2 south *A*. For the objects of the two outer groups, B_1 and B_3, the exact direction relation is one of the special relations, i.e. B_1 southwest *A* and B_3 southeast *A*. Assuming a uniform distribution of the representative points of the *B* objects and assuming that *B* intersects *A* holds, the exact direction relation of the B objects is distributed as follows: 25% southwest, 25% southeast, 50% south. Generalizing this observation, we find that each of the four generalized (specialized) directions is the exact direction relation for $1/6 \cdot f$ ($1/12 \cdot f$) out of the f neighbors of some source object. Figure 5 illustrates the distribution of the exact direction relations of the *B* objects.

$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$
$\frac{1}{6}$	***A***	$\frac{1}{6}$
$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$

objects *B* satisfying

B* intersects *A

Figure 5. Distribution of the exact direction relations

Under the above assumptions, we can calculate the number of all *starlike* neighborhood paths of a certain length l for a given fanout f of the neighborhood graph. The following lemma gives the order of the number of these paths for $f = 6$ and $f = 12$.

Lemma 2: Let *A* be a spatial object and let l be an integer. Let *intersects* be chosen as the neighborhood relation. If the representative points of all spatial objects are uniformly distributed and if they have the same Δx and Δy, then the number of all *starlike* neighborhood paths with source *A* having a length of at most l is $O(2^l)$ for $f = 12$ and $O(l)$ for $f = 6$.

Lemma 2 allows us to estimate the number of neighborhood paths created when using the filter *starlike*. The assumptions of this lemma may seem to be too restrictive for real applications. Note, however, that *intersects* is a very natural neighborhood relation for spatially extended objects. To evaluate the assumptions of uniform distribution of the representative points of the spatial objects and of the same size of these objects, we conducted a set of experiments to compare the expected numbers of neighborhood paths with the actual number of paths created on a real database.

A geographic database on Bavaria was used for this experimental evaluation. The database contains the ATKIS 500 data [Atk 96] and the Bavarian part of the statistical data obtained by the German census of 1987. Also included are spatial objects representing natural object like mountains or rivers and infrastructure such as highways or railroads. The total number of spatial objects in the database is n = 6,924 and the database size is 57.4 MB.

The average number f of edges of a node plays a crucial role in lemma 2. This lemma calculates the number of starlike neighborhood paths for values of $f = 6$ and $f = 12$. Therefore, we created a different neighborhood graph for each of these f values from the same geographic database by using the neighborhood relations $distance_{\leq a}$ and $distance_{\leq b}$. The distances a and b were set such that the resulting f value was 6 and 12 respectively, that is the total number of edges e was $e = 6 * n$ and $e = 12 * n$ respectively. In our test database, we found $f \approx 6$ for the neighborhood relation *intersect* implying that the above distance a was close to 0. We selected typical communities from the geographic database as source objects according to the following criteria. The communities should be located sufficiently far enough from the Bavarian border so that neighborhood paths with a length of at least 5 can be created. There should be a balanced mix of small and large communities (in terms of their area) since the number of actual neighbors of a community is correlated to its area.

We created *all* neighborhood paths as well as the *starlike* neighborhood paths with a maximum length of up to 5 for each of the selected sources. Table 1 reports the results for $f = 6$ and for $f = 12$. The table shows the results depending on the parameter maximum length. The largest value of maximum length was only 5 due to the very large number of *all* neighborhood paths and the corresponding large runtime for creating them. Note that the numbers presented in the columns “all paths“ and “starlike paths“ do only count the number of paths having a length of exactly the specified *max-length*, i.e. they do not count the shorter paths. The columns “factor of increase” give the

quotient of the number of paths in the current row and the number of paths in the previous row (i.e. for the previous value of *max-length*).

	fanout *f* = 6				**fanout *f* = 12**			
max-length	all paths	factor of increase (all paths)	starlike paths	factor of increase (starlike)	all paths	factor of increase (all paths)	starlike paths	factor of increase (starlike)
1	4	-	4	-	4	-	4	-
2	42	10.5	42	10.5	77	19.3	77	19.3
3	275	6.6	68	1.6	991	12.9	214	2.8
4	1,689	6.1	53	0.8	12,469	12.6	331	1.6
5	9,342	5.5	35	0.7	149,760	12.0	347	1.1

Table 1: Numbers of neighborhood paths

We find that for $f = 6$ the number of *all* neighborhood paths (starting from the same source) with a length of at most *max-length* is $O(6^{max\text{-}length})$ and the number of the *starlike* neighborhood paths only grows approximately linear with increasing *max-length* - as stated by lemma 2. For f = 12 the number of *all* neighborhood paths with a length of at most *max-length* is $O(12^{max\text{-}length})$ as we can expect from the lemma. However, the number of the *starlike* neighborhood paths is significantly less than the stated value $O(2^{max\text{-}length})$. This deviation from lemma 2 can be explained as follows. The lemma assumes the same size of the spatial objects. However, small destination objects are more likely to fulfil the filter starlike than large destination objects implying that the size of objects on starlike neighborhood paths tends to decrease. Thus, the factor of increase decreases significantly because in general small objects have less neighbors than large objects. Note that lemma 2 nevertheless yields an upper bound for the number of starlike neighborhood paths created.

The factors of increase, listed in table 1, provide some interesting insights. The factors of increase are approximately as stated by lemma 2. However, we observe that these factors are exceptionally large for *max-length* = 2, i.e. when comparing the paths for *max-length* = 1 and *max-length* = 2. The reason is that the filter starlike does not yield any restrictions for the extension of paths with length 1 since these paths do not yet have a characteristic direction. Therefore, the factor of increase for *max-length* = 2 is the same for all paths as for the starlike paths.

3 Algorithms for Spatial Data Mining

To support our claim that the expressivity of our spatial data mining primitives is adequate, we demonstrate how typical spatial data mining algorithms can be integrated with a spatial DBMS by using the database primitves introduced in section 2.

3.1 Spatial Clustering

Clustering is the task of grouping the objects of a database into meaningful subclasses (that is, clusters) so that the members of a cluster are as similar as possible whereas the members of different clusters differ as much as possible from each other. Applications of clustering in spatial databases are, e.g., the detection of seismic faults by grouping the entries of an earthquake catalog or the creation of thematic maps in geographic information systems by clustering feature spaces.

Different types of spatial clustering algorithms have been proposed, e.g. *k*-medoid clustering algorithms such as CLARANS [NH 94]. This is an example of a *global* custering algorithm (where a change of a single database object may influence all clusters) which cannot make use of our database primitives in a natural way. On the other hand, the basic idea of a *single scan algorithm* is to group neighboring objects of the database into clusters based on a *local* cluster condition performing only one scan through the database. Single scan clustering algorithms are efficient if the retrieval of the neighborhood of an object can be efficiently performed by the SDBS. Note that local cluster conditions are well supported by our database primitives, in particular by the `neighbors` operation on an appropriate neighborhood graph. The algorithmic schema of single scan clustering is depicted in figure 6.

Different cluster conditions yield different notions of a cluster and different clustering algorithms. For example, *GDBSCAN* (*Generalized Density Based Spatial Clustering of Applications with Noise*) [SEKX 98] relies on a density-based notion of clusters. The key idea of a density-based cluster is that for each point of a cluster its *Eps*-neighborhood for some given $Eps > 0$ has to contain at least a minimum number of points, i.e. the “density” in the *Eps*-neighborhood of points has to exceed some threshold. This idea of “density-based clusters” can be generalized in two important ways. First, any notion of a neighborhood can be used instead of an *Eps*-neighborhood if the definition of the neighborhood is based on a binary predicate which is symmetric and reflexive. Second, instead of simply counting the objects in a neighborhood of an object other measures to

```
SingleScanClustering(Database db; NRelation rel)
    set Graph to create_NGraph(db,rel);
    initialize a set CurrentObjects as empty;
    for each node O in g do
        if O is not yet member of some cluster then
            create a new cluster C;
            insert O into CurrentObjects;
            while CurrentObjects not empty do
                remove the first element of CurrentObjects as O;
                set Neighbors to neighbors(Graph, O, TRUE);
                if Neighbors satisfy the cluster condition do
                    add O to cluster C;
                    add Neighbors to CurrentObjects;
end SingleScanClustering;
```

Figure 6. Schema of single scan clustering algorithms

define the "cardinality" of that neighborhood can be used as well. Whereas a distance-based neighborhood is a natural notion of a neighborhood for point objects, it may be more appropriate to use topological relations such as *intersects* or *meets* to cluster spatially extended objects such as a set of polygons of largely differing sizes. See [SEKX 98] for a detailed discussion of suitable neighborhood relations for different applications.

3.2 Spatial Characterization

The task of *characterization* is to find a compact description for a selected subset of the database. In this section, we discuss the task of characterization in the context of spatial databases and review two relevant methods.

Extending the general concept of association rules, [KH 95] introduces *spatial association rules* which describe associations between objects based on spatial neighborhood relations. For instance, a user may want to discover the spatial associations of towns in British Columbia with roads, waters, mines or boundaries having some specified support and confidence. Then, the following spatial association rule may be discovered:

$\forall X \in DB \exists Y \in DB$: is-a(X,town) $\rightarrow$ close-to(X,Y) $\wedge$ is-a(Y,water) (80%)

This rule states that 80% of the selected towns are close to water, i.e. the rule characterizes towns in British Columbia as generally being close to some lake, river etc.

The algorithm presented in [KH 95] to find spatial association rules consists of 5 steps. Step 2 (coarse spatial computation) and step 4 (refined spatial computation) involve spatial aspects of the objects and are briefly examined in the following. Step 2 computes spatial joins of the object type to be characterized (such as town) with each of the other specified object types (such as water, road, boundary or mine) using a neighborhood relation (such as close-to). For each of the candidates obtained from step 2 (and which passed an additional filter-step 3), the exact spatial relation, for example *overlap*, is determined in step 4. Finally, a relation such as the one depicted in figure 7 results which is the input for the final step of rule generation. It is easy to see that the spatial steps 2 and 4 of this algorithm can be well supported by the `neighbors` operation on a suitable neighborhood graph.

Town	Water	Road	Boundary
Saanich	<meet, J.FucaStrait>	<overlap,highway1>, <close-to,highway17>	<close-to,US>
PrinceGeorge		<overlap, highway97>	
Petincton	<meet,OkanaganLake>	<overlap, highway97>	<close-to,US>
. . .	. . .	. . .	. . .

Figure 7. Input for the step of rule generation [KH 95]

[EFKS 98] introduces the following definition of spatial characterization with respect to a database and a set of target objects which is a subset of the given database. A *spatial characterization* is a description of the spatial and non-spatial properties which are typical for the target objects but not for the whole database. The relative frequencies of the non-spatial attribute values and the relative frequencies of the different object types are used as the interesting properties. For instance, different object types in a geographic database are communities, mountains, lakes, highways, railroads etc. To obtain a *spatial* characterization, not only the properties of the target objects, but also the properties of their neighbors (up to a given maximum number of edges in the relevant neighborhood graph) are considered.

A spatial characterization rule of the form *target* $\Rightarrow p_1\ (n_1, freq\text{-}fac_1) \wedge ... \wedge p_k\ (n_k, freq\text{-}fac_k)$ means that for the set of all targets extended by n_i neighbors, the property p_i is $freq\text{-}fac_i$ times more (or less) frequent than in the database. The characterization algorithm usually starts with a small set of target objects, selected for instance by a condition on some non-spatial attribute(s) such as "rate of retired people = HIGH" (see figure 8, left). Then, the algorithms expands regions around the target objects, simultaneously selecting those attributes of the regions for which the distribution of values differs significantly from the distribution in the whole database (figure 8, right).

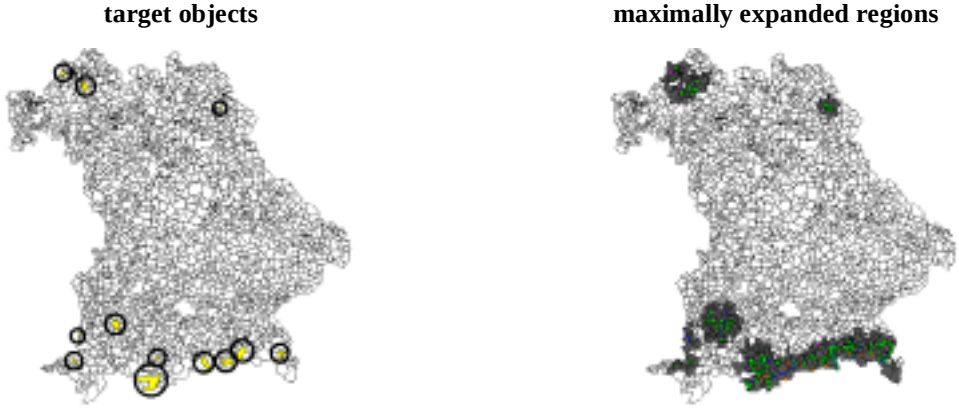


Figure 8. Characterizing wrt. high rate of retired people [EFKS 98]

In the last step of the algorithm, the following characterization rule is generated describing the target regions. Note that this rule lists not only some non-spatial attributes but also the neighborhood of mountains (after three extensions) as significant for the characterization of the target regions:

```
community has high rate of retired people ⇒
      apartments per building = very low (0, 9.1) ∧
      rate of foreigners = very low (0, 8.9) ∧
      rate of academics = medium (0, 6.3) ∧
      average size of enterprises = very low (0, 5.8) ∧
      object type = mountain (3, 4.1)
```

Obviously, this algorithm is well suited for support by the proposed database primitives.

3.3 Spatial Classification

The task of *classification* is to assign an object to a class from a given set of classes based on the attribute values of this object. In *spatial classification* the attribute values of neighboring objects are also considered.

The algorithm presented in [KHS 98] works as follows: The relevant attributes are extracted by comparing the attribute values of the target objects with the attribute values of their nearest neighbors. The determination of relevant attributes is based on the concepts of the *nearest hit* (the nearest neighbor belonging to the same class) and the *nearest miss* (the nearest neighbor belonging to a different class). In the construction of the decision tree, the neighbors of target objects are not considered individually. Instead, so-called *buffers* are created around the target objects and the non-spatial attribute values are aggregated over all objects contained in the buffer. For instance, in the case of shopping malls a buffer may represent the area where its customers live or work. The size of the buffer yielding the maximum information gain is chosen and this size is applied to compute the aggregates for all relevant attributes. Figure 9 depicts an example of a spatial decision tree classifying stores as having a high or low profit.

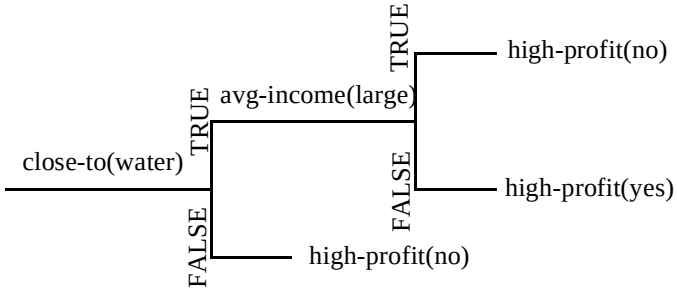


Figure 9. Spatial decision tree [KHS 98]

Whereas the nearest neighbor cannot be directly expressed by our neighborhood relations, it would be possible to extend our set of neighborhood relation accordingly. The proposed database primitives are, however, sufficient to express the creation of buffers for spatial classification.

3.4 Spatial Trend Detection

A *spatial trend* has been defined as a regular change of one or more non-spatial attributes when moving away from a given start object *o* [EFKS 98]. Neighborhood paths starting from *o* are used to model the movement and a regression analysis is performed on the respective attribute values for the objects of a neighborhood path to describe the regularity of change. For the regression, the distance from *o* is the independent variable and the difference of the attribute values are the dependent variable(s) for the regression. The correlation of the observed attribute values with the values predicted by the regression function yields a measure of confidence for the discovered trend.

Global as well as *local* trends are possible. The existence of a global trend for a start object *o* indicates that if considering all objects on all paths starting from *o* the values for the specified attribute(s) *in general* tend to increase (decrease) with increasing distance. Figure 10 (left) depicts the result of algorithm global-trend for the attribute “average rent” and the city of Regensburg as a start object. Algorithm *local-trends* detects single paths starting from an object *o* and having a certain trend. The paths starting from *o* may show different pattern of change, e.g., some trends may be positive while the others may be negative. Figure 10 (right) illustrates this case for the attribute “average rent” and the city of Regensburg as a start object.

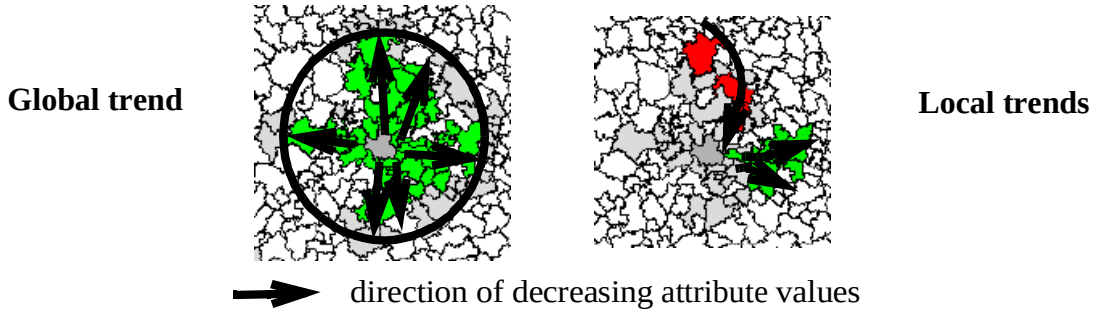


Figure 10. Spatial trends of the“average rent” starting from the city of Regensburg

The algorithms for spatial trend detection are naturally supported by our paths and extensions operation.

4 Efficient DBMS Support Based on Neighborhood Indices

Typically, spatial index structures, e.g. R-trees [Gut 84], are used in an SDBMS to speed up the processing of queries such as region queries or nearest neighbor queries [Gue 94]. Note that our default implementation of the `neighbors` operations uses an R-tree. If the spatial objects are fairly complex, however, retrieving the neighbors of some object this way is still very time consuming due to the complexity of the evaluation of neighborhood relations on such objects. Furthermore, when creating all neighborhood paths with a given source object, a very large number of `neighbors` operations has to be performed. Finally, many SDBS are rather static since there are not many updates on objects such as geographic maps or proteins. Therefore, materializing the relevant neighborhood graphs and avoiding to access the spatial objects themselves may be worthwhile. This is the idea of the neighborhood indices.

4.1 Neighborhood Indices

In this section, we review related work on join indices and then we introduce our concept of neighborhood indices. The idea of a (relational) *join index* [Val 87] is to maintain a precomputed structure containing all pairs of tuples from the two input relations satisfying some join predicate. [Val 87] shows how such indices can be used by a query optimizer to speed-up the processing of join operations. The join index is implemented as a binary relation. [Rot 91] introduced the concept of *spatial join indices* as a materialization of a spatial join with the goal of speeding up spatial query processing. Given two sets of vertices V_1 and V_2 and a set of edges E, an abstract join index is defined as the bipartite graph (V_1, V_2, E). [Rot 91] describes an algorithm to generate a spatial join index from a grid file. In this case, the elements of the V_i represent the page regions, that is the sets of cells of the directory mapped to the same data page. E contains an edge for each pair of vertices from V_1 and V_2 where the corresponding page regions have an ε-overlap for some ε specified by the database administrator. Furthermore, an algorithm is presented for updating the join index on updates of the underlying grid files. [Rot 91] does not discuss the physical design of spatial join indices.

[LH 92] refines the concept of spatial join indices. The elements of the V_i represent objects instead of page regions. A *distance associated join index* consists of tuples of the form ($object_1$,$object_2$,distance($object_1$,$object_2$)) for each pair of database objects. This join index can be used to support not only queries concerning a single neighborhood relation but it is applicable to a large number of queries. Since a distance associated join index requires $O(n^2)$ space for a database of n objects, a hierarchical version is also proposed. These indices assume a spatial concept hierarchy of objects. A *hierarchical distance associated join index* has one entry only for pairs of objects contained in the same object of the next higher level of the hierarchy. For instance, only pairs of cities in the same state or pairs of houses in the same city are represented by an index entry. This approach significantly reduces the space requirements but also prevents its application for spatial data mining if a spatial concept hierarchy is either not available or not relevant for the task of mining. For example, in a geographic information system there may be a spatial concept hierarchy of districts > communities > etc. but the influence of communities to their neighborhood is not restricted to communities of the same district. Consequently, we cannot rely on such hierarchies - representing a political viewpoint - for the purposee of supporting spatial data mining.

Our concept of *neighborhood indices* is related to that of the distance associated join indices with the following new contributions:

- A specified maximum distance restricts the pairs of objects represented in a neighborhood index.
- For each of the different types of neighborhood relations (that is distance, direction, and topological relations), the concrete relation of the pair of objects is stored.

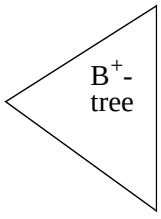
In the following, we introduce neighborhood indices more formally.

Definition 7: neighborhood index

Let *DB* be a set of spatial objects and let *max* and *dist* be real numbers. Let *D* be a direction relation and *T* be a topological relation. Then the *neighborhood index* for *DB* with maximum distance *max*, denoted by I_{max}^{DB}, is defined as follows:

$$I_{max}^{DB} = \{(O_1,O_2,dist,D,T) \mid O_1, O_2 \in DB, O_1\ distance_{=dist}\ O_2 \wedge dist \leq max \wedge O_2\ D\ O_1 \wedge O_1\ T\ O_2\}.$$

A simple implementation of a neighborhood index using a B^+-tree on the key attribute *Object-ID* is illustrated in figure 11.



Object-ID	Neighbor	Distance	Direction	Topology
o_1	o_2	2.7	southwest	disjoint
o_1	o_3	0	northwest	overlap
. . .	. . .	. . .	. . .	. . .

Figure 11. Sample Neighborhood Index

A neighborhood index supports not only one but a set of neighborhood graphs. We call a neighborhood index *applicable* for a given neighborhood graph if the index contains an entry for each of the edges of the graph. To find the neighborhood indices applicable for some neighborhood graph, we introduce the notion of the critical distance of a neighborhood relation *r*. Intuitively, the *critical distance* of a neighborhood relation *r* is the maximum possible distance for a pair of objects O_1 and O_2 satisfying $O_1\ r\ O_2$. The following definitions introduce these notions formally.

Definition 8: applicable neighborhood index

Let G_r^{DB} be a neighborhood graph and let I_{max}^{DB} be a neighborhood index. I_{max}^{DB} is *applicable* for G_r^{DB} iff $\forall(O_1 \in DB, O_2 \in DB) O_1 r O_2 \Rightarrow (O_1, O_2, dist, D, T) \in I_{max}^{DB}$

Definition 9: critical distance of a neighborhood relation

Let r be a neighborhood relation and let $\Re$ denote the set of the real numbers. Let 2^{Points} denote the set of all spatial objects. The *critical distance* of r, denoted as *c-distance*(r), is defined as follows:

$$c\text{-}distance(r) = \begin{cases} min(D) & \text{if } D \text{ is non-empty} \\ \infty & \text{otherwise} \end{cases}$$

with the set D defined as:

$$D = \{d \in \Re \mid \forall (O_1, O_2 \in 2^{Points})(O_1 r O_2 \wedge O_1 distance_{=\,dist} O_2 \Rightarrow dist \leq d)\}$$

The following lemma allows to calculate the critical distance for any neighborhood relation. The critical distance is calculated recursively along the composition of a neighborhood relation.

Lemma 3: The following equation holds for the critical distance of a neighborhood relation r:

$$c\text{-}distance(r) = \begin{cases} 0 & \text{if } r \text{ is a topological relation except } \textit{disjoint} \\ c & \text{if } r \text{ is the relation } distance_{<c} \text{ or } distance_{=c} \\ \infty & \text{if } r \text{ is a direction relation, the relation } distance_{>} \text{ or } \textit{disjoint} \\ min(c\text{–}distance(r_1), c\text{–}distance(r_2)) & \text{if } r = r_1 \wedge r_2 \\ max(c\text{–}distance(r_1), c\text{–}distance(r_2)) & \text{if } r = r_1 \vee r_2 \end{cases}$$

A neighborhood index with a maximum distance of *max* is applicable for a neighborhood graph with relation r if the critical distance of r is not larger than *max*. This is the contents of lemma 4.

Lemma 4: Let G_r^{DB} be a neighborhood graph and let I_{max}^{DB} be a neighborhood index.

If $max \geq c\text{-}distance(r)$, then I_{max}^{DB} is applicable for G_r^{DB}.

Obviously, if two neighborhood indices I_{c1}^{DB} and I_{c2}^{DB} with $c_1 < c_2$ are available and applicable, using I_{c1}^{DB} is more efficient because in general it has less entries than I_{c2}^{DB}. The *smallest applicable neighborhood index* for some neighborhood graph is the applicable neighborhood index with the smallest critical distance.

In figure 12, we sketch the algorithm for processing the `neighbors` operation which makes use of the smallest applicable neighborhood index. If there is no applicable neighborhood index, then the standard approach of using some spatial index structure is followed.

```
neighbors (graph G_r^DB, object o, predicate pred)
    select as I the smallest applicable neighborhood index for G_r^DB ; // Index Selection
    if such I exists then                                              // Filter Step
        use the neighborhood index I to retrieve as candidates the set of objects c
        having an entry (o,c,dist, D, T) in I
    else use the spatial index for DB to retrieve as candidates
        the set of objects c satisfying o r c;
    initialize an empty set of neighbors;                              // Refinement Step
    for each c in candidates do
        if o r c and pred(c) then
            add c to neighbors
    return neighbors;
```

Figure 12. Algorithm neighbors

The first step of algorithm neighbors, the *index selection*, selects a neighborhood index. The *filter step* returns a set of candidate objects (which may satisfy the specified neighborhood relation) with a cardinality significantly smaller than the database size. In the last step, the *refinement step*, for all these candidates the neighborhood relation as well as the additional predicate *pred* are evaluated and all objects passing this test are returned as the resulting neighbors.

To implement the `extensions` operation, we perform a depth-first search. Thus, a path buffer of size *O*(*max-length)* is sufficient to store the intermediate results. On the other hand, a breadth-first search would require a much larger buffer size since it begins creating all paths before finishing the first one. To retrieve the nodes for potential extensions of a candidate path, the `neighbors` operations is used indicating that the efficiency of this operation is crucial. Figure 13 presents the algorithm for the `extensions` operation in pseudo-code notation.

To create a neighborhood index I_{max}^{DB}, a spatial join on *DB* with respect to the neighborhood relation $(O_1 distance_{= dist} O_2 \wedge dist \leq max)$ is performed. A spatial join can be efficiently processed by using a spatial index structure, see e.g. [BKSS 94]. For each pair of objects returned by the spatial join, we then have to determine the exact distance, the direction relation and the topological relation. The resulting tuples of the form $(O_1, O_2, Distance, Direction, Topology)$ are stored in a relation which is indexed by a B-tree on the attribute O_1.

```
extensions(graph g, list of paths p, integer max-length, filter f)
    initialize an empty list extensions;
    initialize the list of path-candidates to the list p;
    while path-candidates is not empty do
        remove the first element of path-candidates as cand;
        if length of cand < max-length then
            set o to last node of cand;
            call neighbors(g,o,TRUE) obtaining the set neighborhood;
            for each element neighbor of neighborhood do
                create an extension ext of cand by adding neighbor as the last node;
                if ext is valid and ext satisfies the filter f then
                    add ext to extensions;
                    add ext at the head of the list path-candidates;
    return path-candidates;
```

Figure 13. Algorithm extensions

Updates of a database, i.e. insertions or deletions of spatial objects, require updates of the derived neighborhood indices. Fortunately, the update of a neighborhood index I_{max}^{DB} is restricted to the neighborhood of the respective object defined by the neighborhood relation $A\ distance_{< \max}\ B$. This neighborhood can be efficiently retrieved by using either a neighborhood index (in case of a deletion) or by using a spatial index structure (in case of an insertion). As an example, we discuss *insertions* of a new spatial object *o* to a database of spatial objects *d*. The retrieval of the relevant neighbors of *o* is not supported by any neighborhood index since *o* is a new object. However, the spatial index structure assumed to be available for *d* supports this retrieval. Note that there may be several neighborhood indices derived from the same database *d* and that all relevant ones have to be updated to include an entry for each of the neighbors of *o*. Figure 14 presents the algorithm *insert-object* in pseudo-code notation. In each of the relevant neighborhood indices, two entries have to be inserted for each pair (*o*,*neighbor*). Recall that the direction relations and the topological relations are not symmetric so that the relation *r* has to be determined for *o r neighbor* as well as for *neighbor r o*.

insert-object(database *d*, object *o*)
  set *maximum* to the maximum distance of the largest neighborhood index
    $I^d_{maximum}$ derived from *d*;
  retrieve as *neighborhood* all objects *n* from *d* satisfying $dist(n, o) \leq maximum$
    by using the spatial index structure associated with *d*;
  for each element *neighbor* of *neighborhood* **do**
    calculate *distance* as dist(*neighbor*,*o*);
    calculate *direction* as the direction relation of *neighbor* and *o*;
    calculate *reverse-direction* as the direction relation of *o* and *neighbor*;
    calculate *topology* as the topological relation of *neighbor* and *o*;
    calculate *reverse-topology* as the topological relation of *o* and *neighbor*;
    for each neighborhood index I^d_{max} derived from *d* **do**
      if $distance \leq max$ **then**
        insert the entry (*neighbor, o,distance,direction, topology*) into I^d_{max};
        insert the entry (*o, neighbor, distance,reverse-direction,*
          reverse-topology) into I^d_{max};

Figure 14. Algorithm insert-object

4.2 Cost Model

A cost model is developed to predict the cost of performing a `neighbors` operation with and without a neighborhood index. For database algorithms, usually the number of page accesses is chosen as the cost measure. However, the amount of CPU time required for evaluating a neighborhood relation on spatially extended objects such as polygons may become very large so that we model both, the I/O time and the CPU time for an operation. We use t_{page}, i.e. the execution time of a page access, and t_{float}, i.e. the execution time of a floating point comparison, as the units for I/O time and CPU time, respectively.

In table 2, we define the parameters of the cost model and list typical values for each of them. The system overhead *s* includes client-server communication and the overhead induced by several SQL queries for retrieving the relevant neighborhood index and the minimum bounding box of a polygon (necessary for the access of the R-tree). p_{index} and p_{data} denote the probability that a requested index page and data page, respectively, have to be read from disk according to the buffering strategy.

name	meaning	typical values
n	number of nodes in the neighborhood graph	$[10^3 . . 10^5]$
f	average number of edges per node in the graph	$[1 . . 10^2]$
v	average number of vertices of a spatial object	$[1 . . 10^3]$
ff	ratio of fanout of the index and fanout (f) of the graph	[1 . . 10]
c_{index}	capacity of a page in terms of index entries	128
c_v	capacity of a page in terms of vertices	64
p_{index}	probability that a given index page must be read from disk	[0..1]
p_{data}	probability that a given data page must be read from disk	[0..1]
t_{page}	average execution time for a page access	$1 * 10^{-2}$ sec
t_{float}	execution time for a floating point comparison	$3 * 10^{-6}$ sec
s	system overhead	depends on the DBMS

Table 2: Parameters of the cost model

Table 3 shows the cost for the three steps of processing a `neighbors` operation with and without a neighborhood index. In the R-tree, there is one entry for each of the n nodes of the neighborhood graph whereas the B+-tree stores one entry for each of the $f * n$ edges. We assume that the number of R-tree paths to be followed is proportional to the number of neighboring objects, i.e. proportional to f. A spatial object with v vertices requires v/c_v data pages. We assume a distance relation as neighborhood relation requiring v^2 floating point comparisons. When using a neighborhood index, the filter step returns $ff * f$ candidates. The refinement step has to access their index entries but does not have to access the vertices of the candidates since the refinement test can be directly performed by using the attributes *Distance*, *Direction* and *Topology* of the index entries. This test involves a constant, i.e. independent of v, number of floating point comparisons and requires no page accesses such that its cost can be neglected.

4.3 Experimental Results

We implemented the database primitives on top of the commercial DBMS Illustra [Ill 97] using its 2D spatial data blade which provides R-trees. A geographic database of Bavaria was used for an experimental performance evaluation and validation of the cost model. This database represents

Step	Cost without neighborhood index	Cost with neighborhood index
Index Selection	s	s
Filter Step	$f \cdot \lceil \log_{c_{index}} n \rceil \cdot p_{index} \cdot t_{\text{page}}$	$\lceil \log_{c_{index}} (f \cdot n) \rceil \cdot p_{index} \cdot t_{\text{page}}$
Refinement Step	$(1 + f) \cdot \lceil v / c_v \rceil \cdot p_{data} \cdot t_{\text{page}} + f \cdot v^2 \cdot t_{\text{float}}$	$ff \cdot f \cdot p_{data} \cdot t_{\text{page}}$

Table 3: Cost model for the `neighbors` operation

the Bavarian communities with one spatial attribute (polygon) and 52 non-spatial attributes (such as average rent or rate of unemployment). All experiments were run on HP9000/715 (50MHz) workstations under HP-UX 10.10.

The first set of experiments compared the performance predicted by our cost models with the experimental performance when varying the parameters *n*, *f* and *v*. The results show that our cost model is able to predict the performance reasonably well. For instance, figure 15 depicts the results for n = 2,000, v = 35 and varying values for *f*.

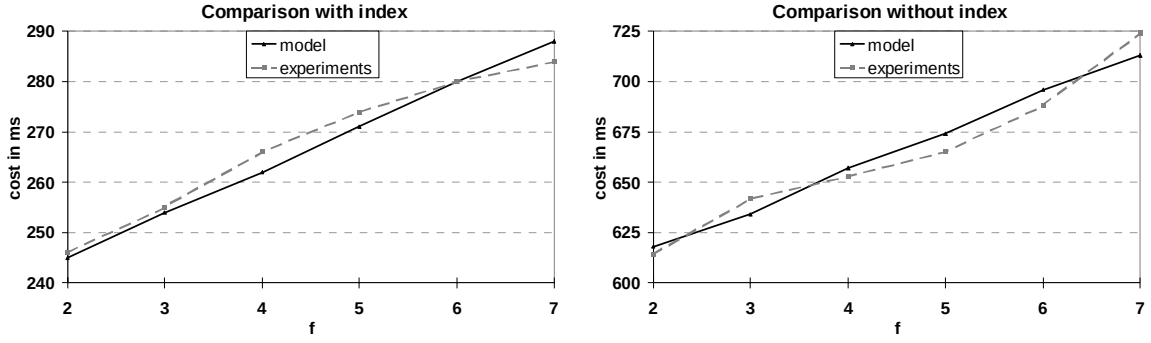


Figure 15. Comparison of cost model versus experimental results

We used our cost model to compare the performance of the `neighbors` operation with and without neighborhood index for combinations of parameter values which we could not evaluate experimentally with our database. Figure 16 depicts the results (1) for f = 10, v = 100 and varying n and (2) for n = 100,000, f = 10 and varying v. These results demonstrate a significant speed-up for the `neighbors` operation with compared to without neighborhood index. In particular, the neighborhood index is very efficient for complex spatial objects, i.e. for large values of v which is typical, e.g., for geographic information systems.

The next set of experiments analyzed the system overhead which is rather large for a single `neighbors` operation. This overhead, however, can be reduced when calling multiple correlated

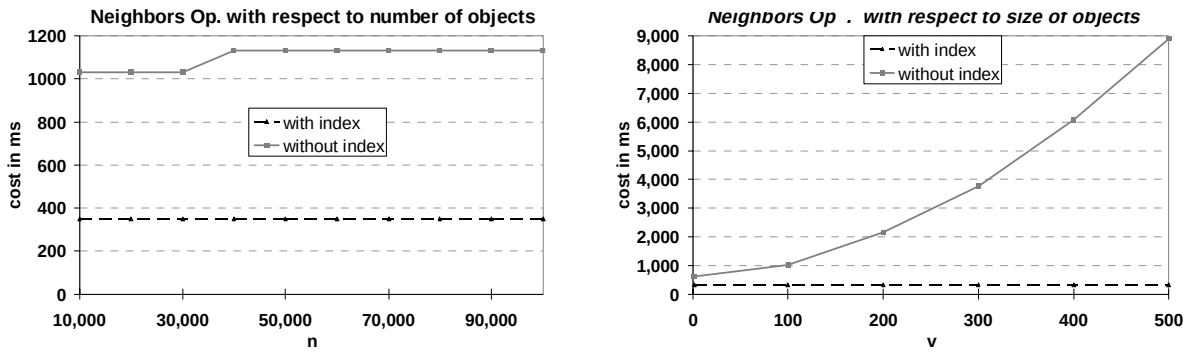


Figure 16. Comparison with and without neighborhood index

neighbors operations issued by one `extensions` operation, since the client-server communication, the retrieval of the relevant neighborhood index etc. is necessary only once for the whole `extensions` operation and not for each of the `neighbors` operations. In our experiments, we found that the system overhead was typically reduced by 50%, e.g. from 211 to 100 ms.

To conclude, when using neighborhood indices we obtain a significant speed-up for the `neighbors` operation. This operation is most crucial to the efficient DBMS support of the database primitives since the implementation of the operations for extending neighborhood paths is based on the `neighbors` operation. The speed-up grows strongly with increasing number of vertices of the spatial objects. There is a large system overhead induced by the DBMS which is significantly reduced when considering sets of `neighbors` operations issued from the same `extensions` operation.

5 Conclusions

In this paper, we defined neighborhood graphs and paths and a small set of database primitives for spatial data mining. We discussed filters restricting the search to such neighborhood paths "leading *away*" from a starting object. An analytical as well as an experimental analysis demonstrated that in typical applications the exponential number of all neighborhood paths can be reduced to a linear number of relevant neighborhood paths. We showed that spatial data mining algorithms such as spatial clustering, characterization, classification and trend detection are well supported by the proposed operations.

Finally, we introduced neighborhood indices to speed-up the processing of our database primitives. Neighborhood indices can be easily created in a commercial DBMS by using standard func-

tionality, i.e. relational tables and index structures. We implemented the database primitives on top of the object-relational DBMS Illustra. The efficiency of the neighborhood indices was evaluated by using an analytical cost model and an extensive experimental study on a geographic database. The implementation using neighborhood indices yielded a significant speed-up compared to the standard implementation using a spatial index structure such as the R-tree. The speed-up grows strongly with increasing complexity of the spatial objects.

Future research includes the following issues. So far, the neighborhood relations between two objects depend only on the properties of the two involved objects. In the future, we will extend our approach to neighborhood relations such as “being among the *k*-nearest neighbors” which depend on more than the two related objects. The investigation of other filters for neighborhood paths with respect to their effectiveness and efficiency in different applications is a further interesting issue. Finally, there are several issues of optimizing the implementation of the database primitives. The system overhead induced by the DBMS is significantly influenced by the “tightness” of the integration of the primitives with the DBMS. A lot of communication overhead could be avoided by implementing the database primitives in the core of the DBMS server and not as a client process. The internal system overhead can be further reduced when using a DBMS such as the Informix Universal Server that allows preparing sets of SQL queries (issued by similar neighbors operations) such that an execution plan is generated only once. In fact, we are currently porting our implementation from Illustra to the Informix Universal Server.

References

[AIS 93] Agrawal R., Imielinski T., Swami A.: *“Database Mining: A Performance Perspective”*, IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 6, 1993, pp. 914-925.

[BF 91] Bill, Fritsch: “*Fundamentals of Geographical Information Systems: Hardware, Software and Data*” (in German), Wichmann Publishing, Heidelberg, Germany, 1991.

[Ege 91] Egenhofer M. J.: *“Reasoning about Binary Topological Relations”*, Proc. 2nd Int. Symp. on Large Spatial Databases, Zurich, Switzerland, 1991, pp. 143-160.

[EKS 97] Ester M., Kriegel H.-P., Sander J.: *“Spatial Data Mining: A Database Approach”*, Proc. 5th Int. Symp. on Large Spatial Databases, Berlin, Germany, 1997, pp. 47-66.

[EFKS 98] Ester M., Frommelt A., Kriegel H.-P., Sander J.: *"Algorithms for Characterization and Trend Detection in Spatial Databases"*, Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, New York City, NY, 1998, pp. 44-50.

[FPS 96] Fayyad U. M., .J., Piatetsky-Shapiro G., Smyth P.: *"From Data Mining to Knowledge Discovery: An Overview"*, in: Advances in Knowledge Discovery and Data Mining, AAAI Press, Menlo Park, 1996, pp. 1 - 34.

[Gue 94] Gueting R. H.: *"An Introduction to Spatial Database Systems"*, Special Issue on Spatial Database Systems of the VLDB Journal, Vol. 3, No. 4, October 1994.

[Gut 84] Guttman A.: *"R-trees: A Dynamic Index Structure for Spatial Searching"*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47-54.

[Ill 97] Informix Inc.: *"Illustra User's Guide"*, Version 3.3, 1997.

[KH 95] Koperski K. and Han J.: "*Discovery of Spatial Association Rules in Geographic Information Databases*", Proc. 4th Int. Symp. on Large Spatial Databases (SSD '95), Portland, ME, 1995, pp 47-66.

[KHA 96] Koperski K., Adhikary J., Han J.: *"Knowledge Discovery in Spatial Databases: Progress and Challenges"*, Proc. SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Technical Report 96-08, University of British Columbia, Vancouver, Canada, 1996.

[KHS 98] Koperski K. , Han J., Stefanovic N.: *"An Efficient Two-Step Method for Classification of Spatial Data"*, Proc. Symposium on Spatial Data Handling (SDH '98), Vancouver, Canada, 1998.

[LH 92] Lu W., Han J.: *"Distance-Associated Join Indices for Spatial Range Search"*, Proc. 8th Int. Conf. on Data Engineering, Phoenix, AZ, 1992, pp. 284-292.

[NH 94] Ng R. T., Han J.:*"Efficient and Effective Clustering Methods for Spatial Data Mining"*, Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, 1994, pp. 144-155.

[Rot 91] Rotem D.: "*Spatial Join Indices"*, Proc. 7th Int. Conf. on Data Engineering, Kobe, Japan, 1991, pp. 500-509.

[SEKX 98] Sander J., Ester M., Kriegel H.-P., Xu X.: *"Density-Based Clustering in Spatial Databases: A New Algorithm and its Applications"*, Data Mining and Knowledge Discovery, an International Journal, Kluwer Academic Publishers, Vol.2, No. 2, 1998.

[Val 87] Valduriez P.: "*Join Indices*", ACM Transactions on Database Systems, Vol. 12, No. 2, 1987, pp. 218-246.