

Invited Paper at 23rd German Conf. on Artificial Intelligence (KI '99),
Bonn, Germany, 1999.

Knowledge Discovery in Spatial Databases

Martin Ester, Hans-Peter Kriegel, Jörg Sander

Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 Muenchen, Germany
{ester | kriegel | sander}@dbs.informatik.uni-muenchen.de
<http://www.dbs.informatik.uni-muenchen.de>

Abstract. Both, the number and the size of spatial databases, such as geographic or medical databases, are rapidly growing because of the large amount of data obtained from satellite images, computer tomography or other scientific equipment. Knowledge discovery in databases (KDD) is the process of discovering valid, novel and potentially useful patterns from large databases. Typical tasks for knowledge discovery in spatial databases include clustering, characterization and trend detection. The major difference between knowledge discovery in relational databases and in spatial databases is that attributes of the neighbors of some object of interest may have an influence on the object itself. Therefore, spatial knowledge discovery 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. Thus, providing general concepts for neighborhood relations as well as an efficient implementation of these concepts will allow a tight integration of spatial knowledge discovery algorithms with a spatial database management system. This will speed-up both, the development and the execution of spatial KDD algorithms. For this purpose, we define a small set of database primitives, and we demonstrate that typical spatial KDD algorithms are well supported by the proposed database primitives. By implementing the database primitives on top of a commercial database management system, we show the effectiveness and efficiency of our approach, experimentally as well as analytically. The paper concludes by outlining some interesting issues for future research in the emerging field of knowledge discovery in spatial databases.

1 Introduction

Knowledge discovery in databases (KDD) has been defined as the process of discovering valid, novel, and potentially useful patterns from data [9]. *Spatial Database Systems (SDBS)* (see [10] 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 [12] for an overview).

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. Similar to the relational standard query 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. [1] follows a similar approach for mining in relational databases. Our database prim-

itives (section 2) are based on the concept of neighborhood relations. The proposed primitives are sufficient to express most of the algorithms for spatial data mining from the literature (section 3). We present techniques for efficiently supporting these primitives by a DBMS (section 4). Section 5 summarizes the contributions and discusses several issues for future research.

2 Database Primitives for Spatial Data Mining

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 (see [7] for a first sketch) are based on the concept of spatial neighborhood relations.

2.1 Neighborhood Relations

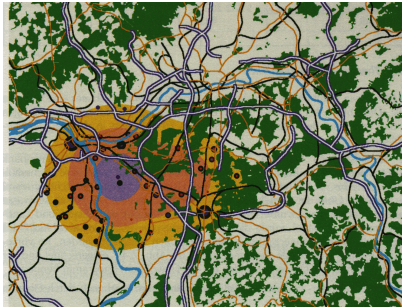


Fig. 1. Regions of pollution around a planned industrial plant [3]

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 industrial plant. Furthermore, the influenced objects such as communities and forests are depicted.

communities and forests are depicted.

We introduce three basic types of binary spatial relations: topological, distance and direction relations. 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 $p = (p_1, p_2, \dots, p_d)$ 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 x - and the y -dimension.

Topological relations are relations which are invariant under topological transformations, i.e. they are preserved if both objects are rotated, translated or scaled simultaneously.

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 definition can be found in [5].

Distance relations are those relations comparing the distance of two objects with a given constant using one of the arithmetic operators.

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 A and B be spatial objects, i.e. $A, B \in 2^{Points}$. Then a distance relation $A \text{ distance}_{\sigma c} B$ holds iff $dist(A, B) \sigma c$.

In the following, we define 2-dimensional direction relations and we will use their geographic names. We define the direction relation of two spatially extended objects using one representative point $rep(A)$ of the *source* object A and all points of the *destination* object B . The representative point of a source object is used as the origin of a virtual coordinate system and its quadrants define the directions.

Definition 3: (direction relations) Let $rep(A)$ be the representative of a 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 .

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. 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 (*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 spatial objects.

a) A *neighborhood graph* $G_{neighbor}^{DB} = (N, E)$ is a graph where the set of nodes N corresponds to the set of objects $o \in DB$. The set of edges $E \subseteq N \times N$ contains the pair of nodes (n_1, n_2) 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 $[n_1, n_2, \dots, n_k]$, 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.

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 $n * f^{k-1}$.

Obviously, the number of 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. Therefore, the operations on neighborhood paths will provide parameters (filters) to further reduce the number of paths actually created.

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. Furthermore, we define a small set of basic operations on neighborhood graphs and paths as database primitives for spatial data mining. In this paper, we introduce only the two most important of these operations:

`neighbors: NGraphs x Objects x Predicates --> 2Objects`
`extensions: NGraphs x 2NPaths x Integer x Predicates -> 2NPaths`

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`. 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.

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. 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 one possible filter predicate, i.e. *starlike*. Other filters may be useful depending on the application.

Definition 6: (filter starlike) Let $p = [n_1, n_2, \dots, n_k]$ 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 $k=1$.



Fig. 2. 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*.

Under the following 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.

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 extension in both x and y direction, 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 $f = 6$. (see [6] for a proof)

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 from a real geographic database on Bavaria. The database contains the ATKIS 500 data [2] and the Bavarian part of the statistical data obtained by the German census of 1987.

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 $O(6^{max-length})$ 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 $O(12^{max-length})$ as we can expect from the lemma. However, the number of the *starlike* neighborhood paths is significantly less than $O(2^{max-length})$. This effect can be explained as follows. The lemma assumes equal 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. Note that lemma 2 nevertheless yields an upper bound for the number of *starlike* neighborhood paths created.

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 expressed by the database primitives 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 group-

ing 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. 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 the `neighbors` operation on an appropriate neighborhood graph. The algorithmic schema of single scan clustering is depicted in figure 3.

```

SingleScanClustering(Database db; NRelation rel)
  set Graph to create_NGraph ( db , rel );
  initialize a set CurrentObjects as empty;
  for each node O in Graph 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;

```

Fig. 3. Schema of single scan clustering algorithms

Different cluster conditions yield different notions of a cluster and different clustering algorithms. For example, *GDBSCAN* [16] 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 ϵ -neighborhood has to contain at least a minimum number of points. 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 ϵ -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 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.

3.2 Spatial Characterization

The task of *characterization* is to find a compact description for a selected subset (the *target set*) of the database. A *spatial characterization* [8] 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 $target \Rightarrow p_1(n_1, freq-fac_1) \wedge \dots \wedge p_k(n_k, freq-fac_k)$ means that for the set of all targets extended by n_i neighbors, the property p_i is $freq-fac_i$ times more (or less) frequent than in the whole 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 4, left). Then, the algorithm 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 4, right).

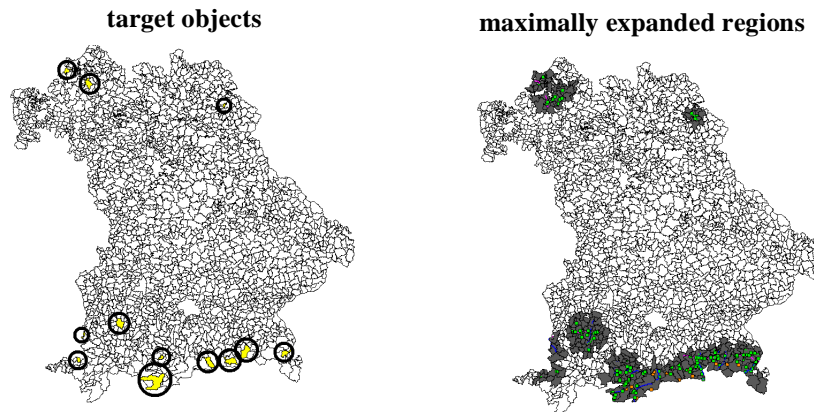


Fig. 4. Characterizing wrt. high rate of retired people [8]

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  $\Rightarrow$ 
  apartments per building = very low (0, 9.1)  $\wedge$ 
  rate of foreigners = very low (0, 8.9)  $\wedge$ 
  . . .  $\wedge$ 
  object type = mountain (3, 4.1)
```

3.3 Spatial Trend Detection

A *spatial trend* [8] is as a regular change of one or more non-spatial attributes when moving away from a given start object o . 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.

Algorithm *global-trends* detects global trends around a start object o . 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 5 (left) depicts the result of algorithm *global-trends* for the trend attribute “average rent” and a start object representing the city of Regensburg.

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, for example, some trends may be positive while the others may be negative. Figure 5 (right) illustrates this case again for the trend attribute “average rent” and the start object representing the city of Regensburg.

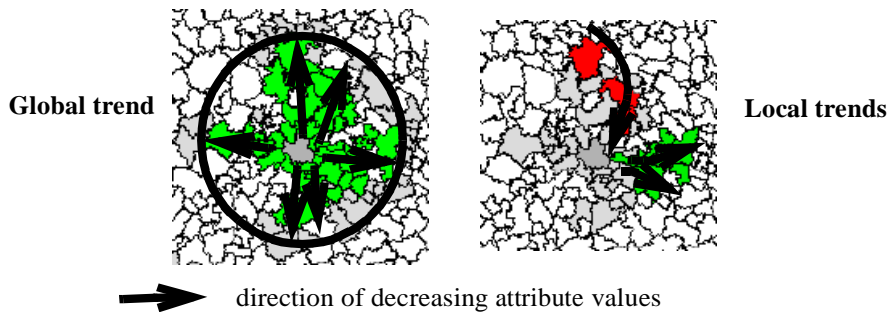


Fig. 5. Trends of the “average rent” starting from the city of Regensburg

4 Efficient DBMS Support Based on Neighborhood Indices

Typically, spatial index structures, e.g. R-trees [11], are used in an SDBMS to speed up the processing of queries such as region queries or nearest neighbor queries [10]. Therefore, 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

Our concept of *neighborhood indices* is related to the work of [15] and [13]. [15] introduced the concept of *spatial join indices* as a materialization of a spatial join with the goal of speeding up spatial query processing. This paper, however, does not deal with the questions of efficient implementation of such indices. [13] extends spatial join indices by associating each pair of objects with their distance. In its basic form, this index requires $O(n^2)$ space because it needs one entry not only for pairs of neighboring objects but for each pair of objects. Therefore, in [13] a hierarchical version of distance

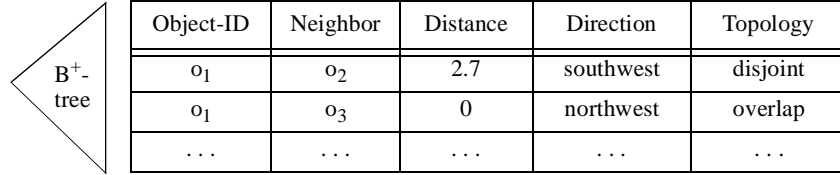
associated join indices is proposed. In general, however, we cannot rely on such hierarchies for the purpose of supporting spatial data mining. Our approach, called *neighborhood indices*, extends distance associated join indices with the following new contributions:

- 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.

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: $I_{max}^{DB} = \{(O_1, O_2, dist, D, T) \mid$

$$O_1, O_2 \in DB \wedge O_1 \text{ distance}_{=dist} O_2 \wedge dist \leq max \wedge O_2 D O_1 \wedge O_1 T O_2\}.$$

A simple implementation of a neighborhood index using a B^+ -tree on the key attribute *Object-ID* is illustrated in figure 6.



Object-ID	Neighbor	Distance	Direction	Topology
o ₁	o ₂	2.7	southwest	disjoint
o ₁	o ₃	0	northwest	overlap
...

Fig. 6. 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. 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 $O_1 r O_2$.

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. The *critical distance* of r , denoted by $c\text{-distance}(r)$, is defined as follows:

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

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 .

Lemma 3: 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 $I_{c_1}^{DB}$ and $I_{c_2}^{DB}$ with $c_1 < c_2$ are available and applicable, using $I_{c_1}^{DB}$ is more efficient because in general it has less entries than $I_{c_2}^{DB}$. The *smallest applicable neighborhood index* for some neighborhood graph is the applicable neighborhood index with the smallest critical distance.

In figure 7, 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 that uses an R-tree 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 R-tree 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;

```

Fig. 7. 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`. The `extensions` operation can obviously be implemented by iteratively performing `neighbors` operations. Therefore, it is obvious that the performance of the `neighbors` operation is very important for the efficiency of our approach.

To create a neighborhood index I_{max}^{DB} , a spatial join on DB with respect to the neighborhood relation $(O_1 \text{distance}_{=dist} O_2 \wedge dist \leq max)$ is performed. A spatial join can be efficiently processed by using a spatial index structure, see e.g. [4]. 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 .

Updates of a database, i.e. insertions or deletions, 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 $A \text{distance}_{<max} B$. 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).

4.2 Cost Model

We developed a cost model to predict the cost of performing a `neighbors` operation with and without a neighborhood index. 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 1, 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.

Table 1: Parameters of the cost model

name	meaning	typical values
n	number of nodes in the neighborhood graph	$[10^3 \dots 10^5]$
f	average number of edges per node in the graph (fan out)	$[1 \dots 10^2]$
v	average number of vertices of a spatial object	$[1 \dots 10^3]$
ff	ratio of fanout of the index and fanout (f) of the graph	$[1 \dots 10]$
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 DBMS

Table 2 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 implying that its cost can be neglected.

Table 2: Cost model for the `neighbors` operation

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

4.3 Experimental Results

We implemented the database primitives on top of the commercial DBMS Illustra 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 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 8 depicts the results for $n = 2,000$, $v = 35$ and varying values for f .

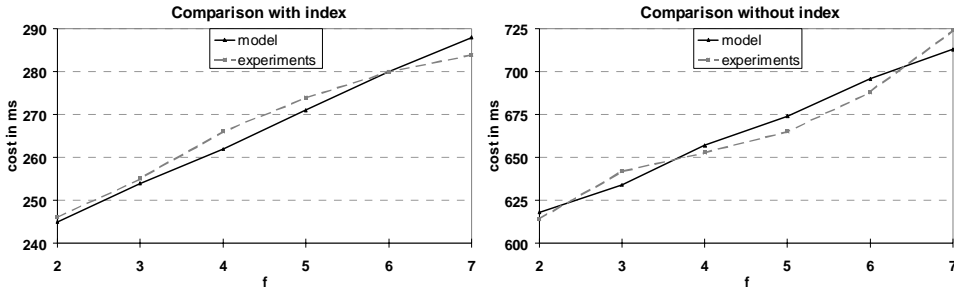


Fig. 8. Comparison of cost model versus experimental results

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 9 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. Furthermore, this speed-up grows strongly with increasing number of vertices of the spatial objects.

The next set of experiments analyzed the system overhead which is rather large. This overhead, however, can be reduced when calling multiple correlated `neighbors` operations issued by one `extensions` operation, since the client-server communication,

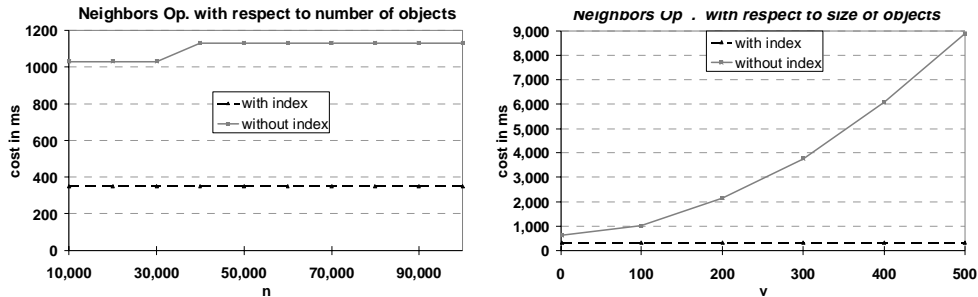


Fig. 9. Comparison with and without neighborhood index

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 ms to 100 ms, when calling multiple correlated `neighbors` operations.

5 Conclusions

In this paper, we defined neighborhood graphs and paths and a small set of database primitives for spatial data mining. We showed that spatial data mining algorithms such as spatial clustering, characterization, and trend detection are well supported by the proposed operations. Furthermore, 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 the effectiveness of the proposed filter. 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 functionality, 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.

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, a tighter integration of the database primitives with the DBMS should be investigated.

References

- [1] 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.
- [2] Bavarian State Bureau of Topography and Geodasy, CD-Rom, 1996.
- [3] Bill, Fritsch: “*Fundamentals of Geographical Information Systems: Hardware, Software and Data*” (in German), Wichmann Publishing, Heidelberg, Germany, 1991.

- [4] Brinkhoff T., Kriegel H.-P., Schneider R., and Seeger B.: “*Efficient Multi-Step Processing of Spatial Joins*”. Proc. ACM SIGMOD '94, Minneapolis, MN, 1994, pp. 197-208.
- [5] Egenhofer M. J.: “*Reasoning about Binary Topological Relations*”, Proc. 2nd Int. Symp. on Large Spatial Databases, Zurich, Switzerland, 1991, pp. 143-160.
- [6] Ester M., Gundlach S., Kriegel H.-P., Sander J.: “*Database Primitives for Spatial Data Mining*”, Proc. Int. Conf. on Databases in Office, Engineering and Science (BTW'99), Freiburg, Germany, 1999.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] Guttman A.: “*R-trees: A Dynamic Index Structure for Spatial Searching*”, Proc. ACM SIGMOD '84, 1984, pp. 47-54.
- [12] 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, UBC, Vancouver, Canada, 1996.
- [13] 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.
- [14] 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.
- [15] Rotem D.: “*Spatial Join Indices*”, Proc. 7th Int. Conf. on Data Engineering, Kobe, Japan, 1991, pp. 500-509.
- [16] 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.