Proc. of the Fifth Int. Symposium on Large Spatial Databases (SSD '97),
Berlin, Germany, Lecture Notes in Computer Science, Springer, 1997.

Spatial Data Mining: A Database Approach

Martin Ester, Hans-Peter Kriegel, Jörg Sander

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

Abstract. Knowledge discovery in databases (KDD) is an important task in spatial databases since both, the number and the size of such databases are rapidly growing. This paper introduces a set of basic operations which should be supported by a spatial database system (SDBS) to express algorithms for KDD in SDBS. For this purpose, we introduce the concepts of neighborhood graphs and paths and a small set of operations for their manipulation. We argue that these operations are sufficient for KDD algorithms considering spatial neighborhood relations by presenting the implementation of four typical spatial KDD algorithms based on the proposed operations. Furthermore, the efficient support of operations on large neighborhood graphs and on large sets of neighborhood paths by the SDBS is discussed. Neighborhood indices are introduced to materialize selected neighborhood graphs in order to speed up the processing of the proposed operations.

Keywords: Spatial Data Mining, Neighborhood Graphs, Efficient Query Processing.

1 Introduction

Spatial Database Systems (SDBS) are database systems for the management of spatial data. Both, the number and the size of spatial databases are rapidly growing in applications such as geomarketing, traffic control and environmental studies. This growth by far exceeds human capacities to analyze the databases in order to find implicit regularities, rules or clusters hidden in the data. Therefore, automated knowledge discovery becomes more and more important in spatial databases. *Knowledge discovery in databases (KDD)* is the non-trivial extraction of implicit, previously unknown, and potentially useful information from databases [FPM 91].

A wide variety of algorithms have been proposed for KDD. [MCP 93] tries to classify these algorithms and identifies the following generic tasks:

- *class identification*, i.e. grouping the objects of the database into meaningful subclasses.
- *classification*, i.e. finding rules that describe the partition of the database into a given set of classes.
- *dependency analysis*, i.e. finding rules to predict the value of some attribute based on the value of another attribute.
- *deviation detection*, i.e. discovering deviations from the expectations, e.g. outliers in a class of objects.

While a lot of algorithms have been developed for KDD in relational databases, the area of KDD in spatial databases has only recently emerged (see [KHA 96] for an overview). The goal of this paper is to define a set of basic operations for KDD in SDBS which can be used to express many relevant algorithms in the sense that most of the relevant queries in a relational database can be expressed using the five basic operations of relational algebra. [AIS 93] follow a similar approach for KDD in relational databases. The definition of such a set of basic operations and their efficient support by an SDBS will speed up both, the development of new spatial KDD algorithms and their performance.

The rest of the paper is organized as follows. Section 2 discusses a sample geographic information system to illustrate the tasks of KDD in SDBS and to motivate the definition of the basic operations. We present the concepts of neighborhood graphs and paths together with their basic operations in section 3. Section 4 demonstrates the applicability of the proposed basic operations by presenting four spatial KDD algorithms based on these operations, two of them from literature and two new ones. Section 5 discusses efficient database support for neighborhood graphs and paths and their operations. Section 6 summarizes the contributions of this paper and discusses several issues for further research.

2 KDD Tasks in a Geographic Information System

A *geographic information system* is an information system for data representing aspects of the surface of the earth together with relevant facilities such as roads or houses. In this section, we introduce a sample geographic database providing spatial and non-spatial information on Bavaria with its administrative units such as communities, its natural facilities such as the mountains and its infrastructure such as roads. We use an extended relational model and the SAND (Spatial And Non-spatial Database) architecture [AS 91]. The spatial extension of the objects (i.e. polygons or lines) is stored and manipulated using an R*-tree [BKSS 90].

A small part of the relation communities is depicted in figure 1. The geographic database BAVARIA may be used, e.g., by economic geographers to discover spatial rules on the economic power of communities. Some non-spatial attribute such as the unemployment rate is chosen as an indicator of the economic power. In a first step, areas with a locally minimal unemployment rate are determined which are called *centers*, e.g. the city of Munich. The theory of central places [Chr 68] claims that the attributes of central cities influence the attributes of their neighborhood in a degree which decreases with increasing distance. E.g., in general it is easy to commute daily from some community to a close by center implying a low unemployment rate in this community. Thus, in a second step the theoretical distribution of the unemployment rate in the neighborhood of the centers is calculated, e.g.

```
when moving away from Munich,
the unemployment rate increases
```

Due to the general assumption of spatial continuity [IS 89], this distribution is typically continuous. In a third step, deviations from the theoretical distribution are discovererd, e.g.

```
when moving away from Munich towards the north east,
the unemployment rate decreases
```

Communities	name	population	unemploy-ment rate	rate of foreigners	spatial extension
	Munich	1.300.000	0.06	0.15	
	...	...	...	...	...

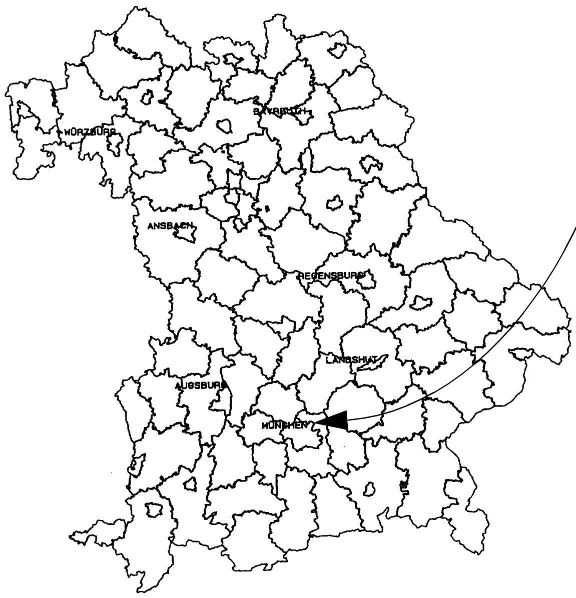


Fig. 1. The communities with their spatial and non-spatial attributes

The goal of the fourth step is to explain these deviations. E.g., if some community is relatively far away from a center but is well connected to it by train, the unemployment rate in this community is not as high as theoretically expected. In our example, the deviation is explained by the location of the Munich airport:

```
when moving away from Munich towards the airport,
the unemployment rate decreases
```

Summarizing, knowledge discovery according to the theory of central places is performed in the following steps:

(1) discover centers, i.e. local extrema of some non-spatial attribute

(2) determine the (theoretical) trend of some non-spatial attribute when moving away from the centers

(3) discover deviations from the theoretical trend

(4) explain the deviations by other spatial objects e.g. by some existing infrastructure in that area.

Another typical approach for knowledge discovery in geographic databases is to find interesting correlations between different characteristics of certain areas. E.g., we might find that areas with a high value for the attribute `rate of retired people` are highly correlated with neighboring mountains and lakes. This KDD task is performed in two steps:

(1) find areas of spatial objects, e.g. clusters or neighboring objects, which are homogeneous with respect to some attribute values.

(2) find associations with other characteristics of these areas, e.g. by correlating them with reference maps or with other attribute values.

We conjecture that these tasks of KDD are representative not only for economic geography but also for a broader class of applications of geographic information systems, e.g. for environmental studies.

Some other approaches to extract knowledge from spatial databases have been proposed in the literature. In [LHO 93] attribute-oriented induction is applied to spatial and non-spatial attributes using (spatial) concept hierarchies to discover relationships between spatial and non-spatial attributes. A spatial concept hierarchy represents a successive merging of neighboring regions into larger regions. In [NH 94] the clustering algorithm CLARANS which groups neigboring objects automatically without a spatial concept hierarchy is combined with attribute-oriented induction on non-spatial attributes. [KH 95] introduces spatial association rules which are discussed in more detail in section 4.1. [Ng 96] and [KN 96] present algorithms to detect properties of clusters using reference maps and thematic maps. E.g. a cluster may be explained by the existence of certain neigboring objects which may “cause” the existence of the cluster.

3 Basic Operations for KDD in SDBS

Roughly speaking, SDBS are relational databases plus a concept of spatial location and spatial extension. The explicit location and extension of objects define implicit relations of spatial neighborhood. We claim that most KDD algorithms for *spatial* databases will make use of those neighborhood relationships, because it is the main difference between KDD in relational DBS and in SDBS, that attributes of the neighbors of some object of interest may have an influence on the object and therefore have to be considered as well. Furthermore, the discussion of the sample applications in the previous section indicates that the efficiency of many KDD algorithms for SDBS depends heavily on an efficient processing of these neighborhood relationships since the neighbors of many objects have to be investigated in a single run of a KDD algorithm.

Therefore, we present in this section a novel approach to KDD in spatial databases aiming at an extension of SDBSs with data structures and operations for efficient processing of implicit relations of spatial neigbhborhoods. This approach allows a tight integration of spatial KDD algorithms with the database management system of a SDBS, speeding up both the development and the execution of spatial KDD algorithms.

3.1 Neighborhood Graphs and Neighborhood Paths

We introduce the concept of neighborhood graphs explicitly representing those implicit neighborhood relations relevant for KDD tasks. [EG 94] follows a similar approach for modeling networks such as roads or telephone lines for the purpose of spatial query processing. While [EG 94] deals with graphs of explicit networks, we use graphs of implicit relations.

A *neighborhood graph* $G_{neighbor}$ for some spatial relation "neighbor" is a graph (N,E) with the set of nodes N and the set of edges E. Each *node* corresponds to an object of the database and two nodes n_1 and n_2 are connected via some *edge* iff neighbor(object(n_1),object(n_2)) holds. The predicate neighbor may be one of the following *neighborhood relations:*

Topological-Relations (c.f. [Ege 91])
e.g. {*meet, overlap, covers, covered-by, contains, inside, equal*}
Metric-Relations e.g. {*distance* < *d*}
Direction-Relations e.g. {*north, south, west, east*}

Based on the neighborhood graphs, we define a *neighborhood path* in some graph *G* as a list of nodes of *G* with an edge of *G* connecting each pair of successors in the list, e.g. $[n_1, n_2, \ldots, n_k]$ where neighbor(n_i, n_{i+1}) holds for each i, $1 \leq i \leq k - 1$. We define the *length* of a path as the number of its nodes.

3.2 The Basic Operations

Now, we present a set of basic operations on neighborhood graphs and paths designed to support KDD tasks such as those discussed in the previous sections. We use the expressions `nRelations`, `nGraphs` and `nPaths` to denote sets of neighborhood relations, neighborhood graphs and neighborhood paths, respectively.

Note that we do not define an explicit domain of Databases. Instead, we use the domain 2^{Objects} of all subsets of the set of all objects to define our operations. We assume that standard operations such as `select(db:Set-Of-Objects;pred:Predicate)` and `get-value(o:Object;attr:Attribute)` are supported by the SDBS. In the following, we introduce the new operations on neighborhood graphs and paths to be provided by an SDBS. We present both, the signature of the operations and a short description of their meaning.

`get_nGraph:` 2^{Objects} `x nRelations -> nGraphs`

The operation `get_nGraph(db,rel)` returns the neighborhood graph representing the neighborhood relation `rel` on the objects of `db`. Note that `rel` may either be one of the primitive neighborhood relations such as `intersects` or a conjunction or disjunction of two neighborhood relations such as `intersects and north`.

`get_neighborhood: nGraphs x Objects x Predicates -> 2^Objects`

The operation `get_neighborhood(graph,o,pred)` returns the set of all objects o_i directly connected to `o` via some edge of `graph` satisfying the conditions expressed by the predicate `pred`. An additional selection condition `pred` is used if we want to investigate only a specific class of neighbors of object `o` or if we want to exclude explicitly certain types of neighbors of `o`. The definition of `pred` may use spatial as well as non-spatial attributes.

`create_nPaths: 2^Objects x nGraphs x Predicates x Int -> 2^nPaths`

The operation `create_nPaths(objects,graph,pred,i)` creates the set of all paths starting from one of the `objects` and following the edges of the neighborhood graph `graph` with `length` $\leq$ `i`. The predicate `pred` expresses further constraints on the paths to be created. This argument of create_nPaths is the most important for the computational complexity of KDD algorithms operating on sets of paths because the number of all paths in a neighborhood graph tends to be very large. Furthermore, most of the neighborhood graphs will contain many cycles because most of the neighborhood predicates are symmetric. However, for the purpose of KDD we are mostly interested in in a certain class of paths, that is to say paths which are "leading *away*" from the starting object in a straightforward sense. We think 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 understandable patterns (if any will be produced at all). Therefore we assume that the predicate `pred` will in general be defined with respect to a path $p=[n_1,n_2,...,n_k]$ like "direction of the edge $(n_i, n_{i+1}) \cong$ direction of the edge (n_{i-1}, n_i)" or "distance$(n_1, n_{i+1}) >$ distance(n_1, n_i)".

`extend: 2^nPaths x nGraphs x Pred x Int -> 2^nPaths`

The operation `extend(set_of_paths,graph,pred,i)` returns the set of all paths extending one of the paths of `set_of_paths` by up to `i` edges of `graph`. The predicate `pred` is assumed to be the same as in the `create_nPaths` operation that was used to create the `set_of_paths`. Note that the members of `set_of_paths` are not contained in the result such that an empty result indicates that none of the elements of `set_of_paths` could be extended.

Finally, we assume some host programming language providing the standard operations on sets of paths and on single paths such as `length(path:NeighborhoodPath)` and iterators such as **`for each`** `path` **`in`** `paths`.

4 Spatial KDD Algorithms Using the Basic Operations

In this section, we illustrate the applicability of the proposed basic operations by presenting four spatial KDD algorithms based on these operations, two of them from literature (section 4.1 and section 4.2) and two new ones (section 4.4 and section 4.3).

4.1 Spatial Association Rules

[KH 95] proposes a method for mining spatial association rules consisting of five steps. Step 2 (coarse spatial computation) and step 4 (refined spatial computation) involve spatial aspects of the objects and thus are examined in the following. Step 2 computes spatial joins of the target object type (e.g. town) with each of the other specified object types (e.g. water, road, boundary and mine) using the neighborhood relation g_close_to. For each of the candidates obtained from step 2 which passed step 3, in step 4 the exact spatial relation is determined. Finally, a relation such as the one depicted in figure 2 (c.f. [KH 95]) results which is the input of the non-spatial step 5.

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

Fig. 2. sample candidates with exact spatial relations

Both of these spatial steps can be implemented using the operations on neighborhood graphs as follows. Step 2 requires several operations to select the objects of the specified object types. Then, several calls of `get_neighborhood` on the selected sets of objects with the predicate g_close_to yield neighborhood graphs and `create_nPaths` on these graphs with `i=2` yields the required pairs of objects. Step 4 is based on the neighborhood graphs for the special neighborhood relations, e.g. `special_graph`. Then, the refined spatial computation for a pair of objects (o_1,o_2) is equivalent to the test whether o_2 is element of the result of `get_neighborhood(special_graph,`o_1`,true`).

4.2 Spatial Clustering

Clustering algorithms group a given set of objects into classes, i.e. clusters, such that objects in one class show a high degree of similarity, while objects in different classes are as dissimilar as possible. In the BAVARIA database (see section 2), e.g., a clustering algorithm can be applied to discover centers of high economic power. Several clustering algorithms for large spatial databases have been designed (e.g. [EKX 95] and [EKSX 96]).

The goal of the algorithm DBSCAN [EKSX 96] is to partition a database into sets of objects, i.e. *clusters*, such that the density of objects inside of each cluster is considerably higher than outside of the cluster. Furthermore, the density within the areas of noise is lower than the density in any of the clusters. DBSCAN discovers all clusters in `db` with a density of at least `MinPts` objects in the `Eps` neighborhood of each object. To find a cluster, DBSCAN uses region queries. Since region queries retrieve a special kind of neighborhood of the center point of the region, DBSCAN can be expressed using some of our proposed basic operations. The neighborhood graph defined by the metric predicate "`distance(Object1,Object2) ≤ Eps`" is created and then the Eps-neighborhood of each point is retrieved using the operation `get_neighborhood(NeighborhoodGraph,Point,true)`.

4.3 Spatial Trend Detection

A *trend* may be defined as a temporal pattern in some time series data such as network alarms or occurrences of recurrent illnesses (c.f. [BC 96]). In an SDBS, we define a *spatial trend* as a pattern of change of some non-spatial attribute (attributes) in the neighborhood of some database object, e.g. "when moving away from Munich, the economic power decreases".

In the following we introduce an algorithm which discovers trends in SDBS starting from some object *o*. In each step, the algorithm computes both the local changes of the specified attribute when moving to the neighbors as well as the distance to these neighbors. A linear regression is applied to these pairs of values (change of attribute value, distance). If the resulting correlation coefficient is larger than a specified threshold, the slope of the resulting linear function is returned as the trend for *o*. If the correlation coefficient is not large enough, no trend is discovered for *o*. Figure 3 depicts a map of the attribute *average rent* from the BAVARIA database. A significant trend can be observed for the city of Munich: the average rent decreases quite regularly when moving away from Munich.

In the following, we present the algorithm for discovering spatial trends in pseudo code notation. It returns all trends of the non spatial attribute `attr` in `db` of length between `min_length` and `max_length` with the neighborhood defined by `pred`. The algorithm incrementally tries to find significant trends of maximal length, i.e. it extends all current neighborhood paths by one step, performs a linear regression on the `attr` values of the last objects on these paths and continues if the correlation coefficient of the current trend is at least equal to `minconf`. Note that the algorithm uses a predicate `similar_direction`, i.e. "direction of the edge $(n_i, n_{i+1}) \cong$ direction of the edge (n_{i-1}, n_i)", to restrict the creation of neighborhood paths.

```
discover_spatial_trends(db:Set_of_Objects;sel:Predicate;
       min_length,max_length:Int;minconf:Real;attr:Attribute;
       pred:NeighborhoodRelation;)
   focus:=select(db,sel);
   graph:=get_nGraph(db,pred);
   for each object in focus do
   all_paths:=create_nPaths({object},graph,
                              similar_direction,min_length);
      local_trends:=EMPTY_LIST;
      trend_list:=EMPTY_LIST;
      correlation:=MAXREAL;
      slope:=MAXREAL;
      new_paths:=extend(all_paths,graph,similar_direction,1);
      current_length:=2;
      current_trend:=NO_TREND;
      while current_length < max_length and new_paths ≠ EMPTY
           and correlation > minconf do
         all_paths:=union(new_paths,all_paths);
         for each path in all_paths do
            last_object:=get_object(path,length(path));
            attr_change:= get_value(object,attr)
            - get_value(last_object,attr);
            distance:=dist(object,last_object);
            insert [attr_change,distance] into list local_trends;
         end for each path in all_paths;
         perform_linear_regression(local_trends,slope,
                                   correlation);
         new_paths:=extend(all_paths,graph,similar_direction,1);
         current_length:= current_length + 1;
         if correlation > minconf then
            current_trend:=[object,slope,correlation];
         end if // correlation > minconf;
      end while // current_length < max_length and . . and . . ;
      if current_trend ≠ NO_TREND then
         insert current_trend into trend_list;
      end if // current_trend ≠ NO_TREND;
   end for // each object in focus;
   return trend_list;
end // discover_trends;
```

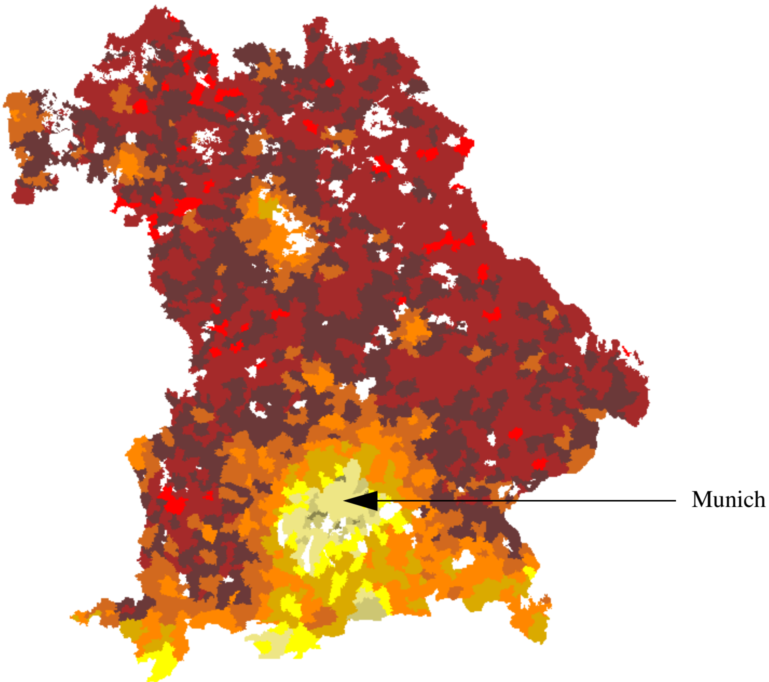


Fig. 3. Average rent for the communities of Bavaria

4.4 Spatial Classification

We assume a database of objects described by a collection of attributes each having a small domain of discrete values. The task of *classification* is to discover a set of classification rules that determine the class of any object from the values of its attributes. A spatial classification algorithm may, e.g., be used to explain the deviations from some discovered or some theoretical spatial trend. The following algorithm is based on the well-known ID3 algorithm [Qui 86] designed for relational DBS. The extension for SDBS is to consider not only attributes of the object o to be classified but to consider also attributes of neighboring objects, i.e. objects of a neighborhood path starting from *o*. Thus, we define a *generalized attribute* for some neighborhood path $p = [o_1, \ldots, o_k]$ as a tuple (attribute-name, index) where index is a valid position in *p* representing the attribute with attribute-name of object o_{index}. The generalized attribute (economic-power,2), e.g., represents the attribute economic-power of some (direct) neighbor of object o_1.

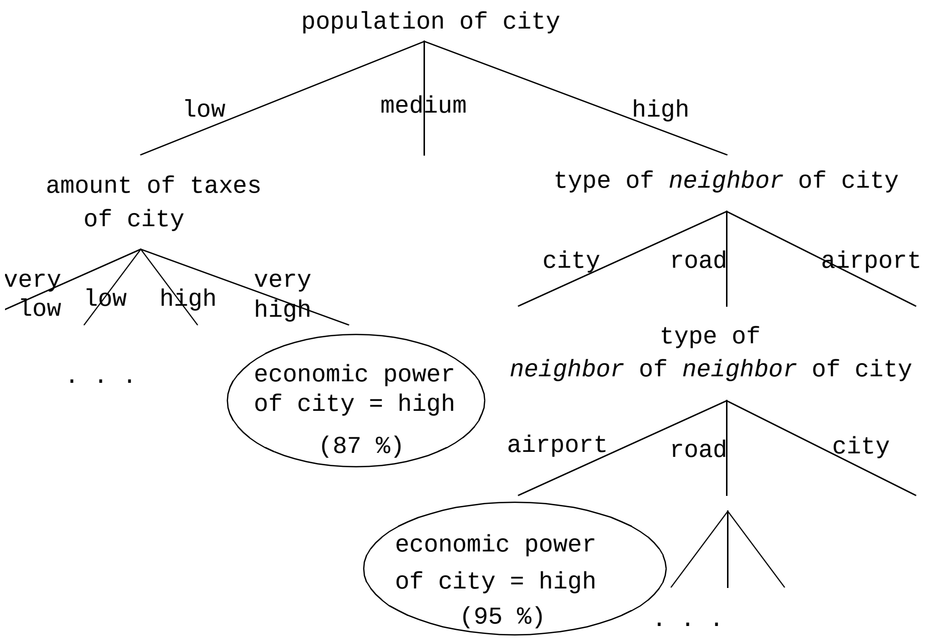
Since the influence of neighboring objects and their attributes decreases with increasing distance, we limit the length of the relevant neighborhood paths by an input parameter max-length. Furthermore, the classification algorithm allows the input of a predicate focusing the search for classification rules on the objects of the database fulfilling this predicate. Figure 4 depicts a sample decision tree and two rules derived from it. `Economic power` has been chosen as the class attribute and the focus is on all objects of type city.

In the following, the algorithm for discovering spatial classification rules is presented in pseudo code notation. It discovers all spatial classification rules, i.e. paths from the root to one of the leaves of the decision tree, with all attributes yielding an information gain of at least ε. Note that the algorithm uses a predicate `larger_distance`, i.e. "distance(n_1, n_{i+1}) > distance(n_1, n_i)" to restrict the creation of neighborhood paths.

```
discover_spatial_classification_rules(db:Set_of_Objects;
      sel:NonSpatialPredicate; class_attr:Attribute;
      pred:NeighborhoodRelation; max_length:Int)
   focus:=select(db,sel);
   neighborhood:=get_nGraph(focus,pred);
   paths:=create_nPaths(focus,neighborhood,larger_distance,
                        max_length);
   classify(class_attr,neighborhood,EMPTY_RULE,paths,
            max_length);
end // discover_classification_rules;

classify(class_attr:Attribute; neighborhood:NeighborhoodGraph;
      rule:ClassificationRule; paths:set_of_paths;
      max_length:Int)
   max_info_gain:=0.0;
   max_attr:=NULL;
   for i from 1 to max_length do
      for each generalized attribute (Aj,i) not used in rule do
         info-gain:=calculate_information_gain(Aj,
                                    class_attr,i,paths);
         if info_gain > max_info_gain then
            max_attr:=Aj;
            max_neighbors:=i;
            max_info_gain:=info_gain;
         end if // info_gain > max_info_gain;
      end for // each attribute Aj not yet used;
   end for // i from 0 to max_length - 1;
   if max_attr ≠ NULL and max_info_gain > ε then
      for each value of max_attr do
         extended_rule:=rule + "max_attr,max_neighbors,value";
         classify(class_attr,neighborhood,
                  extended_rule,paths,max_length);
      end for // each value of max_attr;
   else print rule;
   end if // max_attr ≠ NULL and max_info_gain > ε;
end // classify;
```

```
calculate_information_gain(attr,class_attr: Attribute;
      index:Int;paths:set_of_paths);
   for each path in paths do
      consider attr of the index-th object of path and class_attr
      of the first object of path for the calculation of the
      information gain
   end for // each path in paths;
end // calculate_information_gain;
```



IF population of city = low AND amount of taxes of city = very high
THEN economic power of city = high (87 %)

IF population of city = high AND type of *neighbor* of city = road
AND type of *neighbor* of *neighbor* of city = airport
THEN economic power of city = high (95 %)

Fig. 4. sample decision tree and rules discovered by the classification algorithm

5 Efficient SDBS Support for Neighborhood Graphs and Paths

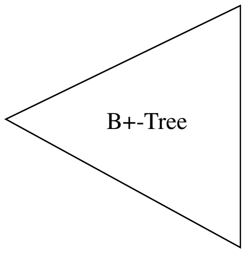
In this section, we discuss the efficient support of the operations on neighborhood graphs and paths by an SDBS. We introduce the concept of neighborhood indices materializing selected neighborhood graphs and show how they can be used to speed up the processing of our basic operations. Furthermore, a cost model is presented which allows to compare the expected execution time of a get-neighborhood operation with vs. without a neighborhood index.

5.1 Neighborhood Indices

First, we discuss some related work. [Rot 91] introduced the concept of *spatial join indices* as a materialization of a spatial join result with the goal of speeding up spatial query processing. This paper, however, does not deal with the questions of efficient implementation of such indices.

[LH 92] extends the concept of spatial join indices by associating their distance with each pair of objects (*distance associated join indices*). Thus, the join index can be used to support not only queries concerning a single spatial predicate but the index is applicable to a large number of queries. In its naive form, however, 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 [LH 92] a hierarchical version of distance associated join indices is proposed. These indices assume a spatial hierarchy of objects, e.g. countries > cities > houses. Entries in the index are only generated for pairs of objects contained in the same object of the next higher level of the hierarchy, e.g. only for pairs of houses of the same city and only for pairs of cities of the same country. The hierarchical approach significantly reduces the space requirements of the join index but also prevents its application to databases if a spatial hierarchy is either not available or the spatial hierarchy is not relevant for the purpose of KDD. E.g. in the geographic information system on Bavaria, there is a spatial hierarchy of districts > cities etc., but the influence of cities, e.g., to their neighborhood is not restricted to cities inside of the same district. Consequently, we cannot rely on such political hierarchies for the purpose of supporting spatial data mining by neighborhood graphs.

In the following, we present our concept of neighborhood indices. We assume the existence of some spatial index such as an R*-tree [BKSS 90] to support spatial query processing. If many operations are performed on the same neighborhood graph and if this graph is relatively stable, an index should be constructed for this neighborhood graph. This seems to be especially important if neighborhood paths are to be constructed from some neighborhood graph. Note that many SDBS are rather static since there are not many updates on objects such as geographic maps or proteins. We define a *neighborhood index* as an index explicitly representing a neighborhood graph, i.e. a neighborhood index supports the processing of all operations on its corresponding neighborhood graph without accessing the database itself. A simple implementation of a neighborhood index using a B+-tree is illustrated in figure 5. In general, a neighborhood graph is undirected implying a double representation of each edge in the neighborhood index.



Object	Neighbors
o_1	o_2, o_3, o_7, o_9
o_2	o_1, o_3, o_9
. . .	. . .

Fig. 5. Sample Neighborhood Index

Clearly, it is prohibitive to construct neighborhood indices for each neighborhood graph. It is the task of the database administrator to select some important neighborhood graphs and to create the neighborhood indices for these graphs. The cost model which is presented in the next section may provide support for this task.

5.2 A Cost Model

A cost model is developed to predict the cost of performing a `get_neighborhood(graph,object,filter)` operation with vs. without a neighborhood index. In the database community, usually the number of page accesses is chosen as the cost measue. However, the amount of CPU time required for evaluating a neighborhood relation on spatially extended objects such as polygons may 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, resp.

In figure 6, we define the parameters of the cost model and list typical values for each of them (see [BKSS 94] for the values of t_{page} and t_{float}):

name	meaning	values
n	number of nodes in the neighborhood graph	$[10^3 \,..\, 10^5]$
e	number of (directed) edges in the neighb. graph	$[10^3 \,..\, 10^6]$
v	average number of vertices of a polygon	$[10 \,..\, 10^4]$
c_{oid}	capacity of a page in terms of object identificators	1000
c_{pol}	average capacity of a page in terms of polygons	4096 / (4 * v)
t_{page}	execution time for a page access	$1 * 10^{-2}$ sec
t_{float}	execution time for a floating point comparison	$3 * 10^{-6}$ sec

Fig. 6. Parameters of the cost model

In a neighborhood index, there is one entry for each of the *e* edges of the associated neighborhood graph. The fan-out of the B+-tree is $c_{oid}/2$ since object identificators are used as keys. On the average, `object` has *e/n* neighbors which have to be read from data pages of the B+-tree. No refinement step has to be performed. Thus, the expected cost for performing a `get_neighborhood(graph,object,filter)` operation with a neighborhood index is as follows:

$$\text{cost with neighborhood index} = \left(\log_{c_{oid}/2} e + \left\lceil \frac{e}{n \times c_{oid}} \right\rceil\right) \times t_{page}$$

If no neighborhood index is available for `graph`, the operation `get_neighborhood(graph,object,filter)` is evaluated directly on the database, i.e. on the R*-tree. In an R*-tree, there is one entry for each of the *n* nodes (i.e. for their associated spatial objects) of the neighborhood graph. The fan-out of the R*-tree is $c_{oid}/5$ (in the case of 2D objects) since bounding boxes are used as keys. We assume that the neighbors of `object` are well clustered on the data pages of the R*-tree and can be read with the minimal number of page accesses. Finally, a refinement step has to be performed. We assume the neighborhood predicate intersects which requires $v * \log v$ floating point comparisons. Hence, the expected cost for performing a `get_neighborhood(graph,object,filter)` operation without using a neighborhood index is as follows:

$$\text{cost without neig. index} = \left(\log_{c_{oid}/5} n + \left\lceil \frac{e}{n \times c_{pol}} \right\rceil\right) \times t_{page} + (v \times \log v) \times t_{float}$$

Using the cost model, we performed a simulation of several scenarios. We considered c_{oid}, c_{pol}, t_{page} and t_{float} to be fixed and *n*, *e* and *v* to be variable. In each scenario, a fixed value is chosen for two of the variable parameters and the third one varies over its domain. The results of the simulation of the three scenarios are depicted in figure 7. The implementation using a neighborhood index significantly outperforms the one without using a neighborhood index. The speed-up increases with increasing values of *e* and *v* and with decreasing values of *n*. To conclude, neighborhood indices are recommended for graphs with a high average number of neighbors (*e/n*) or with large objects (*v*).

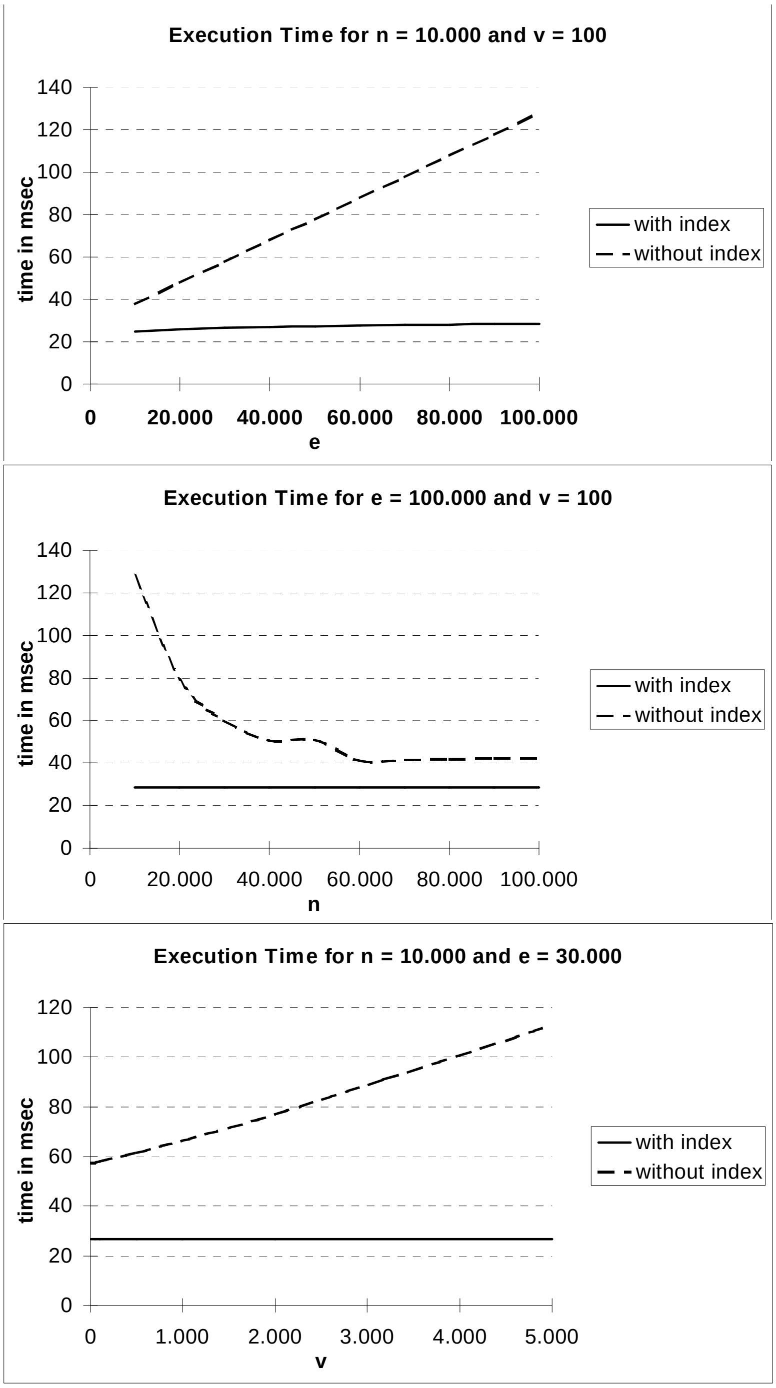


Fig. 7. Simulation of three different scenarios

5.3 Using the Neighborhood Indices

In this section, we discuss the implementation of conjunctions of neighborhood relations, of the operation `create_nPaths` and of updates on a neighborhood index.

So far, we have only considered neighborhood graphs defined by atomic neighborhood predicates. When the graph is defined by a conjunction of neighborhood relations, the neighborhood index of one of the atomic predicates is used to generate candidates which have to be checked further more in a second step. For this purpose, we define the *selectivity* of a neighborhood graph or index as

$$\text{selectivity} \quad = \quad \frac{n}{e}$$

Thus, a high selectivity implies a low average number of neighbors. The implementation of the operation `get_neighborhood(graph,object,filter)` for conjunctions of neighborhood relations follows a two-step approach:

- filter step:
  The most selective of the existing neighborhood indices which apply to the graph is determined. The set of all neighbors of `object` is obtained using this index and is called the `set of candidates.`
- refinement step:
  For all elements of the `set of candidates` it is checked whether the other relations of `graph` hold. This test may either use other neighborhood indices or may be directly performed using the spatial extension of the two objects obtained from the database.

The implementation of the operation `create_nPaths(graph,objects,filter,i)` is based on the operation `get_neighborhood`. A call of `get_neighborhood(graph,last-object,filter)` for the `last-object` of the current path obtains all one-step extensions of the current path. A depth-first traversal of `graph` is more efficient than a breadth-first traversal since the main memory requirements are much smaller. This is due to the fact that the depth-first traversal completes the current path before starting to create another one while the breadth-first traversal has to manage a potentially very large number of incomplete paths before the first one is completed. Redundant calls of `get_neighborhood(graph,object,filter)` for the same arguments can be avoided when providing a buffer of `i` pages so that all neighbors on the current path are always main memory resident. A lot of work on DBS support for path operations has been reported. However, typically the graphs are assumed to be directed and acyclic (cf. [AK 93], [LD 89]) which is not true for neighborhood graphs.

Clearly, updates of a SDB require updates of all its derived neighborhood indices. Fortunately, updates of the neighborhood indices on the insertion of a new object are restricted to the neighbors of the new object in the database. Therefore, the update of the neighborhood index can be efficiently performed using the operation `get_neighborhood`.

6 Conclusions

The main contribution of this paper is the definition of a a set of basic operations for KDD in SDBS which should be supported by an SDBS. The definition of such a set of basic operations and their efficient support by an SDBS will speed up both, the development of new spatial KDD algorithms and their performance. We introduce the concepts of neighborhood graphs and paths and a small set of operations for their manipulation. We argue that these operations are sufficient for KDD algorithms considering spatial neighborhood relations by presenting the implementation of four typical spatial KDD algorithms based on the proposed operations. Two of these algorithms are well-known from literature, the other two algorithms are new and are important contributions to clarify the differences between KDD in relational and in spatial databases. Furthermore, the efficient support of operations on large neighborhood graphs and on large sets of neighborhood paths by the SDBS is discussed. Neighborhood indices are introduced to materialize selected neighborhood graphs in order to speed up the processing of the proposed operations.

There are several issues for further research. First, the algorithms for optimizing the performance of the KDD operations using the available indices have to be evaluated. Second, the materialization of neighborhood paths will be investigated. This seems to be feasible if appropriate filters are used to create reasonably small sets of paths. A materialization of relevant paths may further speed-up the overall performance of KDD tasks because different KDD algorithms may use the same set of paths and each algorithm may scan this set of paths many times. Finally, applications in such domains as geography, biology or CAD will be investigated to insure the practical impact of our approach.

Acknowledgements

We thank Xiaowei Xu, Stefan Gundlach and Alexander Frommelt (Institute of Computer Science, University of Munich) for fruitful discussions on draft versions of this paper. Henning Brockfeld (Institute of Economic Geography, University of Munich) introduced us into the KDD problems of economic geographers and kindly provided the BAVARIA data.

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.

[AK 93] Agrawal R., Kiernan J.: *“An Access Structure for Generalized Transitive Closure Queries”*, Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 429-438.

[AS 91] Aref W.G., Samet H.: *"Optimization Strategies for Spatial Query Processing"*, Proc. 17th Int. Conf. VLDB, Barcelona, Spain, 1991, pp. 81-90.

[BC 96] Berndt D. J., Clifford J.: *"Finding Patterns in Time Series: A Dynamic Programming Approach"*, in Fayyad U., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. (eds.): Advances in Knowledge Discovery and Data Mining, AAAI Press / The MIT Press, 1996, pp. 229-248.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[BKSS 94] Brinkhoff T., Kriegel H.-P., Schneider R., Seeger B.: *'Efficient Multi-Step Processing of Spatial Joins',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, MN, 1994, pp. 197-208.

[Chr 68] Christaller W.: *"Central Places in Southern Germany"*, (in German), Wissenschaftliche Buchgesellschaft, 1968.

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

[EG 94] Erwig M., Gueting R.H.: *"Explicit Graphs in a Functional Model for Spatial Databases"*, IEEE Transactions on Knowledge and Data Engineering, Vol.6, No.5, 1994, pp. 787-803.

[EKX 95] Ester M., Kriegel H.-P., Xu X.: "*Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification*", Proc. 4th Int. Symp. on Large Spatial Databases, Portland, ME, 1995, pp.67-82.

[EKSX 96] Ester M., Kriegel H.-P., Sander J., Xu X.: *"A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise"*, Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, Oregon, 1996, AAAI Press, 1996.

[FPM 91] Frawley W.J., Piatetsky-Shapiro G., Matheus J.: *"Knowledge Discovery in Databases: An Overview"*, in: Knowledge Discovery in Databases, AAAI Press, Menlo Park, 1991, pp. 1-27.

[IS 89] Isaaks E.H., Srivastava R.M.: *"Applied Geostatistics"*, Oxford University Press, New York, 1989.

[KH 95] Koperski K., Han J.: *"Discovery of Spatial Association Rules in Geographic Information Databases"*, Proc. 4th Int. Symp. on Large Spatial Databases, 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.

[KN 96] Knorr E.M., Ng R.T.: *"Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining"*, IEEE Transactions on Knowledge and Data Engineering, Vol.8, No.6, 1996, pp. 884-897.

[LD 89] Larson P.-A., Deshpande V.: *"A File Structure Supporting Traversal Recursion"*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 243-252.

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

[LHO 93] Lu W., Han J., Ooi B.C.: "*Discovery of General Knowledge in Large Spatial Databases*", Proc. Far East Workshop on Geographic Information Systems, Singapore, 1993, pp. 275-289.

[MCP 93] Matheus C.J., Chan P.K., Piatetsky-Shapiro G.: *"Systems for Knowledge Discovery in Databases"*, IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.6, 1993, pp. 903-913.

[Ng 96] Ng R.T.: "*Spatial Data Mining: Discovering Knowledge of Clusters from Maps"*, Proc. SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Technical Report 96-08, University of British Columbia, Vancouver, Canada, 1996.

[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.

[Qui 86] Quinlan J.R.: *Induction of Decision Trees*, Machine learning 1, 1986, pp. 81 - 106.