Proc. Int. Conf. on Databases in Office, Engineering and Science, (BTW '99), Freiburg, Germany, 1999.

Database Primitives for Spatial Data Mining

Martin Ester, Stefan Gundlach, Hans-Peter Kriegel, Jörg Sander

Institute for Computer Science, University of Munich

Oettingenstr. 67, D-80538 München, Germany

email: {ester | gundlach | kriegel | sander}@dbs.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. 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. In particular, *data mining* is the step of applying appropriate algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data.

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).

In [LHO 93], attribute-oriented induction is performed by 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 describe associations between objects based on different spatial *neighborhood relations*. [KN 96] present algorithms to detect properties of clusters using reference maps and thematic maps. For instance, a cluster may be explained by the existence of certain *neigboring objects* which may "cause" the existence of the cluster. New algorithms for spatial classifcation and spatial trend analysis are sketched in [EKS 97] and elaborated in [EFKS 98]. For spatial classification it is important that class membership of a database object is not only determined by its non-spatial attributes but also by the attributes of objects in its *neighborhood*. In spatial trend analysis, patterns of change of some non-spatial attribute(s) in the *neighborhood* of some database object are determined.

We argue that data mining algorithms should be integrated with existing DBMS, i.e. they should not run on separate files but they should run directly on a database. Thus, redundant storage and potential inconsistencies can be avoided. Furthermore, the query operations provided by a DBMS may be used, for example, to select subsets relevant for data mining or to support the user in evaluating the discovered patterns. In this paper, we introduce a set of database primitives for mining in spatial databases (see [EKS 97] for a first outline). These primitives are sufficient to express most of the algorithms for spatial data mining from the literature, in particular they can express the algorithms reviewed above. We present techniques for efficiently supporting these primitives by a DBMS. [AIS 93] follows a similar approach for mining in relational databases. The use of these database primitives will enable the integration of spatial data mining with existing DBMS's and will speed-up the development of new spatial KDD algorithms.

The rest of the paper is organized as follows. In section 2, several types of neighborhood relations are introduced. Based on neighborhood relations, neighborhood graphs and their operations as well as some important filters are defined in section 3. We present our approach to efficiently support the database primitives by a spatial DBMS in section 4. An extensive performance evaluation is presented in section 5. In section 6 a short summary and some directions for future research are given.

2 Neighborhood Relations

Attributes of the neighbors of some object of interest may have an influence on the object itself. 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.

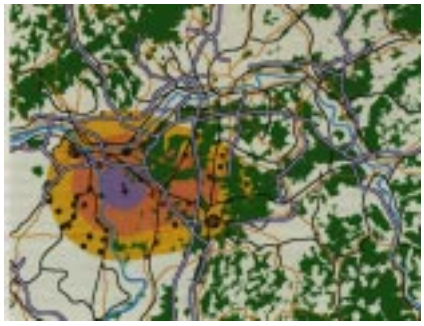


Figure 1. Regions of pollution around a planned industrial plant

We conjecture that most mining algorithms for *spatial* databases will make use of spatial neighborhood relationships, because objects are influenced by the attributes of their neighbors depending on the type of neighborhood relation. 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 point objects 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 or by the points contained in the object, e.g. the pixels of an object in a raster image. Therefore, we use *sets of points* as a generic representation of spatial objects. The points $p = (p_1, p_2, \ldots, p_d)$ are elements of a d-dimensional Euclidean vector space called *Points*. Spatial objects O are represented by a set of points, i.e. $O \in 2^{Points}$. In the case of $d = 2$, $p = (p_x, p_y)$, i.e. p_x denotes the coordinate of p in the first dimension and p_y denotes the coordinate of p in the second dimension. Furthermore, we call $\Delta x(O) := \max\{|o_x - p_x| \mid o, p \in O\}$ 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 definitions of the topological relations 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 *A* and *B* be spatial objects, i.e. $A, B \in 2^{Points}$. Then a distance relation $A\ distance_{\sigma\ c}\ B$ holds iff $dist(A, B)\ \sigma\ c$.

In the following, we define 2D 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 B Rel A*, we distinguish between the *source object A* and the *destination object B* of the direction relation *Rel*. 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 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 the definition of the direction relations using 2D polygons.

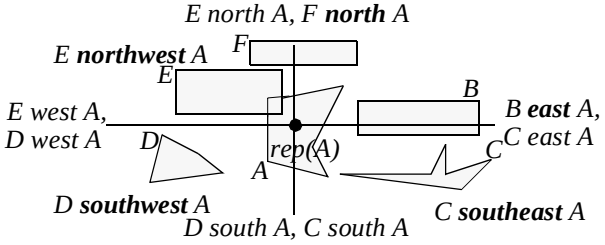


Figure 2. Illustration of the direction relations

Obviously, for each pair of spatial objects one of the direction relations holds but the direction relation between two objects is not uniquely defined. Therefore, we introduce the notion of the exact direction relation of a pair of objects. In figure 2, the exact direction relations are marked by the bold font. This notion makes use of the fact that all the direction relations are partially ordered by a specialization relation.

Definition 4: (**specialization, exact direction relation**)

Let r_1 and r_2 be direction relations. r_2 is a *specialization* of r_1, denoted by $r_2 \{ r_1$, iff

$\forall\ A \in 2^{Points}, B \in 2^{Points}: A\ r_2\ B \implies A\ r_1\ B$.

Let A and B be spatial objects, i.e. $A, B \in 2^{Points}$. r is the *exact direction relation* of A and B iff $\forall\ r^* \in$ DirectionRelations: $A\ r\ B \wedge A\ r^*\ B \Rightarrow r \subseteq r^*$.

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 5: (**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*.

To define neighborhood indices in section 4, we need the notion of the critical distance of a neighborhood relation R, defined inductively along the composition of a neighborhood relation.

Definition 6: (**critical distance of a neighborhood relation**) Let r be a neighborhood relation. The *critical distance* of r, denoted as *c-distance(r)*, is defined as follows:

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

3 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 7: (**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 e denote the cardinality of E. $f := e/n$ denotes the average number of edges of a node, i.e. f is 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 $n_i \in N, 1 \le i < k$. 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}.

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. 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 detection [EFKS 98], 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.Therefore, the operations on neighborhood paths will provide parameters (filters) to further reduce the number of paths actually created and considered in the KDD process.

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), $2^{\texttt{Objects}}$, $2^{\texttt{NPaths}}$. Note that we do not define an explicit domain of databases but we use the domain $2^{\texttt{Objects}}$ of all subsets of the set of all objects.

We assume the standard operations such as *selection*, *union*, *intersection* and *difference* to be available for sets of objects and for sets of neighborhood paths. Therefore, we introduce only the following additional operations:

`neighbors: NGraphs x Objects x Predicates -->` 2^{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.

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

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.

In the following, we discuss two possible filters for neighborhood paths and analyze the expected number of paths created when using these filters.

Definition 8: (**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*) :⇔ ($\exists\ j < k$: $\forall\ j < i < k$: $n_{i+1}\ rel_i\ n_i \Leftrightarrow rel_i \in rel_j$), if $k \geq 3$; TRUE, if $k < 3$

variable-starlike(*p*) :⇔ ($\exists\ j < k$: $\forall\ j < i < k$: $n_{i+1}\ rel_i\ n_i \Leftrightarrow rel_i \in rel_1$), if $k \geq 3$;

TRUE, if $k < 3$. (Note: requires $rel_i \in rel_1$ instead of $rel_i \in rel_j$ in the starlike filter.)

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 least restrictive starlike filter is *variable-starlike*. This filter 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 source object.

Figure 3. Illustration of two different filter predicates

The exact direction relation *rel* of two objects *A* and *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* distinguished by the possible locations of their representative points. Figure 4 illustrates these possible locations for the three groups.

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, the exact direction relation of the *B* objects is distributed as follows: *25% southwest, 25% southeast, 50% south.* Generalizing this observation shows 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

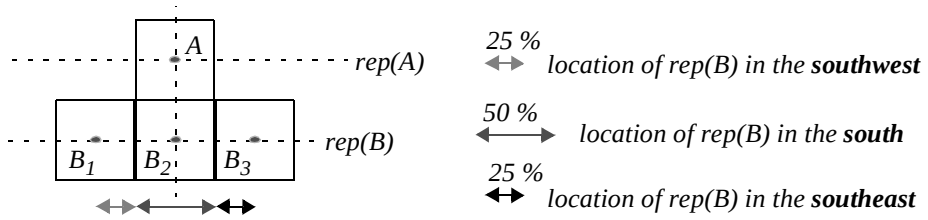


Figure 4. Objects *B* south of *A* and intersecting *A*

object. This leads to the following lemma which shows that the filter *starlike* yields a significant reduction of the numbers of neighborhood paths.

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

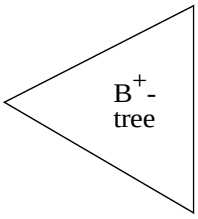
4 Efficient DBMS Support Based on Neighborhood Indices

Typically, spatial index structures, e.g. R-trees [Gut 84], are used in an SDBMS to speed up processing queries such as region queries or nearest neighbor queries [Gue 94]. 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, many SDBS are rather static since there are not many updates on geographic maps or proteins. Therefore, materializing the relevant neighborhood relations and avoiding to access the spatial objects themselves may be worthwhile. This is the idea of the neighborhood index.

This approach is similar to the work of [Rot 91] and [LH 92]. [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. 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 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 [LH 92] 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.

Definition 9: (**neighborhood index**) Let *DB* be a set of spatial objects and let *c* and *dist* be real numbers. Let *D* be a direction predicate and *T* be a topological predicate. Then the *neighborhood index* for *DB* with maximum distance *c*, denoted by N_c^{DB}, is defined as follows: $N_c^{DB} = \{(o_1, o_2, dist, D, T) \mid o_1\, distance_{=dist}\, o_2 \wedge dist \leq c \wedge o_2\, D\, o_1 \wedge o_1\, T\, o_2\}$.

A simple implementation of a neighborhood index using a B^+-tree on the attribute *Object-ID* is illustrated in figure 5.



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

Figure 5. Sample Neighborhood Index

A neighborhood index N_c^{DB} is *applicable* to the neighborhood graph G_r^{DB} if $c \geq$ *c-distance*(*r)* because all neighbors w.r.t. *r* can be found in this neighborhood index. Obviously, if two indices N_{c1}^{DB} and N_{c2}^{DB}, $c_1 < c_2$, are available and applicable, using N_{c1}^{DB} is more efficient because in general it will be smaller than the index N_{c2}^{DB}.

In figure 6 and figure 7 we sketch the algorithms for processing the `neighbors` and the `extensions` operations which use the smallest available index.

neighbors (graph G_r^{DB}, object *o*, predicate *pred*)
• *Index Selection:*
Select the smallest available neighborhood index *N* applicable to G_r^{DB}.
• *Filter Step*:
If *N* exists, use it and retrieve as candidates the neighbors of *o* stored in *N*.
If *N* does not exist, use the spatial index and retrieve as candidates the set of objects *c* satisfying *o r c*.
• *Refinement Step:*
From the set of candidates, return all objects *o'* that fulfill *o r o'* as well as *pred(o')*.

Figure 6. Algorithm neighbors

To implement the `extensions` operation, we perform a depth-first search. Thus, a path buffer of size *max-length* is sufficient to store the intermediate results. 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.

Updates of the database, i.e. insertions, deletions or modifications of spatial objects, require updates of the derived neighborhood indices. Fortunately, these updates are restricted to the neighborhood of the respective object which can be efficiently retrieved by using a `neighbors` operation. Furthermore, typically updates are not very frequent in spatial databases.

```
extensions(graph g, set of paths p, integer max-length, filter f)
    Initialize the list of path candidates to the set p;
    while there is a further candidate of length < max-length do
        remove the first path of candidates as cand;
        Call neighbors(g,o,TRUE) with o=last node of cand
        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
                report the extended path ext as result;
                add ext to the head of the list of path candidates;
            end if;
        end for;
    end while;
```

Figure 7. Algorithm extensions

5 Performance Evaluation

We implemented the database primitives on top of the commercial DBMS Illustra [Ill 97] as a client process. A neighborhood index was realized as a relational table indexed by a B-tree on the Object-ID attribute. Whenever no neighborhood index is applicable, our implementation of the database primitives uses an R-tree which is provided by the Illustra 2D spatial data blade. We developed a cost model to compare the performance with and without neighborhood indices for arbitrary parameter values. 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).

5.1 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 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 several SQL queries to administrative tables 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 1: Parameters of the cost model

Table 2 shows the cost for the three steps (Index Selection, Filter Step and Refinement Step) 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.

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

Table 2: Cost model for the `neighbors` operation

5.2 Experimental Results

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 showed 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*. Note that all costs are measured in ms.

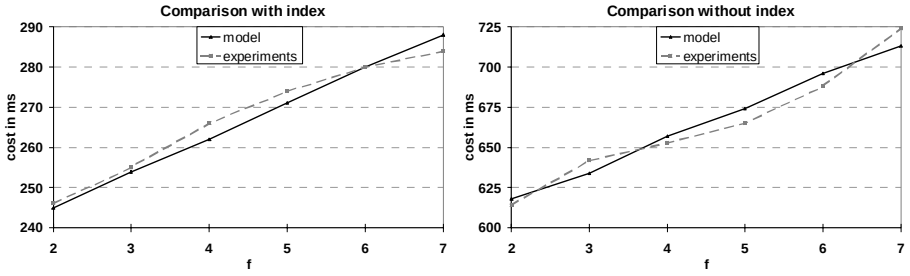


Figure 8. Comparison of cost model versus experimental results

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. In particular, the neighborhood index is very efficient for complex spatial objects, i.e. for large values of v.

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 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.

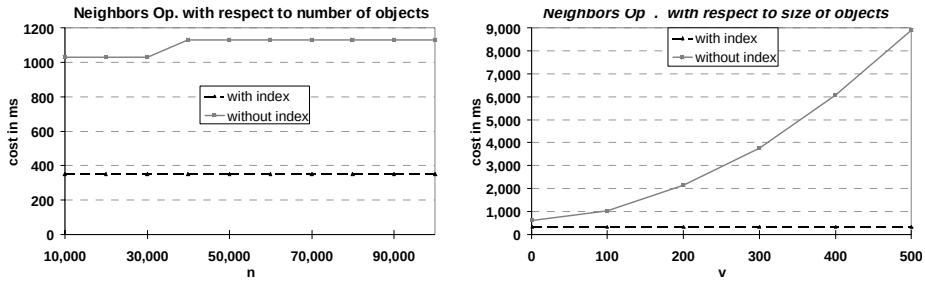


Figure 9. Comparison with and without neighborhood index

The last set of experiments compared the number of all paths with the number of paths created when applying the filter starlike (see figure 10). As predicted by lemma 2, the number of starlike paths is $O(max\text{-}length)$ for $f = 6$ and $O(2^{max\text{-}length})$ for $f = 12$. Furthermore, the neighborhood paths created using this filter proved to be effective for spatial trend detection [EFKS 98].

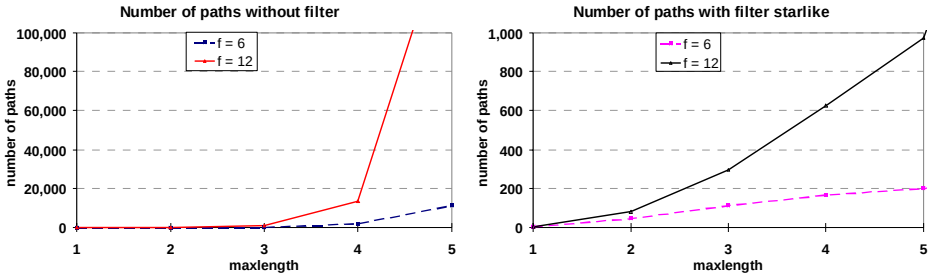


Figure 10. Evaluation of the filter starlike

6 Conclusions

In this paper, we introduced neighborhood graphs and paths and a small set of operations as database primitives for spatial data mining. We discussed two filters restricting the search to such neighborhood paths "leading *away*" from a starting object. Furthermore, we introduced neighborhood indices to support efficient processing of the database primitives by a DBMS. 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.

Future research includes an optimized implementation of the database primitives in the server of a spatial DBMS and a comparison with the current implementation on top of Illustra. Furthermore, other filters should be investigated with respect to their effectiveness and efficiency in different applications. Finally, the proposed database primitives will inspire the development of new spatial data mining algorithms based on neighborhood graphs and paths.

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.

[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] Illustra Information Technologies, Inc.: *"Illustra User's Guide"*, Release 3.3.1 for HP UX 10.1, 1997.

[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.

[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.

[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.

[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.