Efficient Computation of a Proximity Matching in Spatial Databases

Xuemin Lin and Xiaomei Zhou
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW 2052, Australia
lxue@cse.unsw.edu.au

Chengfei Liu
School of Computing Sciences
University of Technology, Sydney
NSW 2009, Australia
liu@socs.uts.edu.au

Abstract

Spatial data mining recently emerges from a number of real applications, such as real-estate marketing, urban planning, weather forecasting, medical image analysis, road traffic accident analysis, etc. It demands for efficient solutions for many new, expensive, and complicated problems. In this paper, we investigate a proximity matching problem among *clusters* and *features*. The investigation involves proximity relationship measurement between clusters and features. We measure proximity in an average fashion to address possible nonuniform data distribution in a cluster. An efficient algorithm is proposed and evaluated to solve the problem. The algorithm applies a standard multi-step paradigm in combining with novel lower and upper proximity bounds. The algorithm is implemented in several different modes. Our experiment results not only give a comparison among them but also illustrate the efficiency of the algorithm.

Keywords: Spatial query processing and data mining.

1 Introduction

Spatial data mining is to discover and understand non-trivial, implicit, and previously unknown knowledge in large spatial databases. It has a wide range of applications, such as demographic analysis, weather pattern analysis, urban planning, transportation management, and etc. While processing of typical spatial queries (such as joins, nearest neighbouring, KNN, and map overlays) has been received a great deal of attention for years [2, 3, 4, 28], spatial data mining, viewed as advanced spatial queries, demands for efficient solutions for many newly proposed, expensive, complicated, and sometimes ad-hoc spatial queries.

Inspired by a success in advanced spatial query processing techniques [2, 3, 4, 11, 12, 28], relational data mining [1, 26, 30], machine learning [9, 10, 22], computational geometry [27], and statistics analysis [17, 29], many research results and system prototypes in spatial data mining have been recently reported [2, 5, 6, 13, 15, 18, 19, 21, 23]. The existing research not only tends to provide system solutions but also covers quite a number of special purpose solutions to ad-hoc mining tasks. The results include efficiently computing spatial association rules [20], spatial data classification and generalization [13, 15, 21, 23], spatial prediction and trend analysis [6], clustering and cluster analysis [5, 7, 18, 25, 32], mining in image and raster databases [8], etc.

Clustering has been shown one of the most useful tools to partition and categorize spatial data into *clusters* for the purpose of knowledge discovery. A number of efficient algorithms [5, 7, 25, 31, 32] have been proposed. Consider that the clustering technique might be too expensive to apply to approaching ad-hoc spatial data mining tasks. In [18, 19], special purpose mining algorithms have been developed, as alternatives to clustering, for solving two ad-hoc problems. The first problem is to find the k closest *features* surrounding a set of points in two dimensional space. Such a set of points may be either a cluster obtained by a clustering algorithm or an existing spatial object (e.g. a residential area) in the database, while a feature is a polygon. The second problem in [18] is to compute the commonality among n sets of points, provided that their k closest features are pre-computed.

To complement the research in [18, 19], in this paper we study the following ad-hoc *proximity matching* problem (PM) among n sets of points:

> Suppose that in two dimensional space, n sets of points and m sets of polygons are given. Regarding to a specific set C of points, find the sets of points that "most proximately" match C with repsect to the following proximity matching condition. For each set C' of points in the solution,
>
> - the "shortest distance" from C' to each set π of polygons is smaller than that between C and π; or
> - if no sets of points meet the above conditition, then the sets that "most closely" meets the above condition are computed.

The PM problem has a number of useful real applications. For instance, in a spatial database for the real-estate development information, a set of points represents a residential area to be developed, where each point represents a land parcel. A polygon corresponds to a vector representation of feature, such as a lake, golf course, school, motor way, etc. In this application, a real-estate developer may want to purchase a well-know residential area C to develop because of its proximity relationships to certain surrounding features but may not be able to do it due to either a budget limit or unavailability. Therefore, the purchaser has to alternatively choose the available and affordable residential areas most similar to C with respect to these proximity relationships. Other applications include road traffic accident investigation, criminal analysis, etc.

PM will be formally defined in the next section. In PM, we assume that the "shortest distance" between a set of points and a set of polygon has not been pre-computed, nor stored in the database. Further, such a short distance will be defined in average sense to reflect non-uniform data distribution [18]. These differentiate PM with KNN [2, 13], the problem of searching commonalities among n sets of points [18], and incremental distance join problem [16].

A naive way to solve PM is to first precisely compute the distance information between each pair of a set of points and a set of polygons, and then to solve the PM problem. However, in practice there may be many sets of points far from being part of solution to PM. Motivated by this, our algorithm adopts a standard multi-step technique [2, 4, 18, 20] in combining with novel and powerful prunine conditions to filter out uninvolved features and sets of points. The algorithm has been implemented in several different modes for performance evaluation. Our experiments clearly demonstrate the efficiency of the algorithm.

The rest of the paper is organized as follows. In section 2, we present a precise definition of PM as well as a brief introduction of an adopted spatial database architecture. Section 3 presents our algorithm for solving PM. Section 4 reports our experimental results. In section 5, a discussion is presented regarding various modifications of our algorithm. This is followed by the conclusions and remarks.

2 Preliminary

In this section we precisely define the PM problem. A feature F is a *simple* and *closed* polygon [27] in the 2-dimensional space. A set C of points in the two dimensional space is called *cluster* for notation simplicity. Following [18], we assume that in PM a cluster is always *outside* [27] a feature. Note that this assumption may support many real applications. For instance, in real-estate data, a cluster represents a set of land parcels, and a feature represents a man-made or natural place of interest, such as lake, shopping center, school, park, and entertainment center. Such data can be found in many electronic maps in a digital library.

To efficiently access large spatial data (usually tera-bytes), in this paper we adopt an extended-relational and a SAND (spatial-and-non-spatial database) architecture [3]; that is, a spatial database consisting of a set of spatial objects and a relational database describing non-spatial properties of these objects. For instance, a set of electronic data describing the development information of residential areas and the feaures information in Sydney metropolitan area may be organized as follows.

- REAL ESTATE (name, status, #lands, average_size, average_price, sub, ..., *g_des*),
- GOLF COURSE (name, #holes, ..., *g_des*),
- SCHOOL (name, type, ... , *g_des*),
- BEACH (name, type, ..., *g_des*).

In the above database schemata, the attribute *g_des* represents a spatial object, which is either a set of points (cluster) or a polygon (feature) in PM. In order to achieve efficient access, in SAND the attribute *g_des* stores only a pointer in the relational table, pointing to the actual spatial object description. For eaxmple, the table REAL ESTATE stores the information of different residential areas. The attribute *status* may take 2 values: 1) N (not available), or 2) A (available). For each residential area, we also have the information, such as the number of land parcels, the average land size, the average price of each land, and in which suburb. Below is an example of PM:

Example 1. *select s.**
from REAL ESATE s, REAL ESTATE s1, GOLF COURSE g,
BEACH w, SCHOOL sc
where s1.name = 'SeaBreez' and s.name $\neq$ *'SeaBreez' and*
s.status ='A' and s.average_price $\leq 400,000$ *and*
g.#holes $= 9$ *and sc.type = 'private'*
proximity-matching *between* $(s.obj, s1.obj)$ *regarding their*
shortest distances to g, sc, and w □

Example 1 is to find a real-estate available to be developed, such that its average land price is less than \$400,000, and its individual shortest distances respectively to the golf courses with 9 holes, to private schools, and to beaches are all smaller than those between the SeaBreez real-estate and the features. If such real-estates do not exist then the ones that most approximately meet the proximity matching conditions will be reported.

Taking the above query as an example, we now formally define PM. In PM, the input consists of:

- a cluster C_0 (e.g. the SeaBreez real-estate in Example 1),
- a set S of clusters (e.g. the available residential areas with average_price not greater than \$400,000 in Example 1),
- a set $\Pi = \{\pi_j : 1 \leq j \leq m\}$ of groups of features (e.g. in Example 1, $m = 3$, π_1 is the set of golf courses with 9 holes, π_2 is the set private schools, and π_3 is the set of beaches).

Given a feature F and a point p outside F, the length of the actual (working or driving) shortest path from p to F is too expensive to compute in the presence of tens of thousands of different roads. In PM, we use the shortest Euclidean distance from p to a point in the boundary of F, denoted by $d(p, F)$, to reflect the geographic proximity relationship between p and F. We believe that on average, the length of an actual shortest path can be reflected by $d(p, F)$. We call $d(p, F)$ the *distance* between p and F. Note that if F degenerates to a point p' then $d(p, p')$ means the Euclidean distance between them; and F may also degenerate to a line. Moreover, for the purpose of computing lower and upper proximity bounds in Section 3, we need to extend the definition of $d(p, F)$ to cover the case when p is inside or on the boundary of F; that is, $d(p, F) = 0$ if p is inside of or on the boundary of F.

A proximity value between a cluster C and a feature F can be defined in a number of ways. We may define it by the shortest distance between the "boundary" of C and the boundary of F. However, as points in C admit an arbitrary distribution, such a proximity value may not be the majority consensus from C; this was shown in [18]. We use the following *average proximity* value to quantitatively model the proximity relationship between F and C:

$$AP(C, F) = \frac{1}{|C|} \sum_{p \in C} d(p, F) \tag{1}$$

Consider that in PM a set π of features normally means a set of the same kind of features. We define the distance between a set π of features and a cluster C to be the smallest average proximity between C and a $F \in \pi$, and it is denoted by $D(C, \pi)$. That is,

$$D(C, \pi) = \min_{F \in \pi} \{AP(C, F)\} \tag{2}$$

As mentioned earlier, if in PM there is no exact matching, then we need to find approximate matchings; in this case, we rank the importance of a set π_j of features by a positive value w_j. More important a feature π_j is, larger the w_j is. The w_j can be assigned by either a user or the system default. Therefore, a set $\{w_j : 1 \leq j \leq m\}$ of positive values is also part of the input of

PM. PM can now be modeled to find the clusters C in S such that the following goal function is minimized.

$$PM_{C_0}(C,\Pi) = \sum_{j=1}^{m} w_j \ pm(D(C,\pi_j), D(C_0,\pi_j)) \tag{3}$$

where,

$$pm(D(C,\pi_j), D(C_0,\pi_j)) = \begin{cases} 0 & \text{if } D(C,\pi_j) \leq D(C_0,\pi_j) \\ D(C,\pi_j) - D(C_0,\pi_j) & \text{otherwise} \end{cases} \tag{4}$$

3 Algorithms for Solving PM

In this section, we present an efficient algorithm for solving PM. The algorithm is denoted by CPM, which stands for **C**omputing the **P**roximity **M**atching.

An immediate way (brute-force) to solve PM is to 1) firstly compute $AP(C,F)$ for each pair of a cluster C and a feature F, 2) secondly compute $D(C,\pi_j)$ for every pair of a C and a π_j, 3) thirdly compute $PM_{C_0}(C,\Pi)$ for each cluster C in S, and 4) finally find the clusters C with the smallest values of $PM_{C_0}(C,\Pi)$. Note that $AP(C,F)$ can be easily computed in $O(|C||F|)$ according to the definition of $AP(C,F)$; and it is the dominant coast. Though the brute-force approach runs in quadratic time regarding the input size, often there may be hundreds of clusters, and tens of thousands features involved in the computation. Moreover, each cluster (feature) may have a number of points (edges). This makes the brute-force approach computationally prohibitive in practice; and our experiment results in Section 4 confirm this.

An alternative way to approach PM is to adopt a multi-step paradigm [2, 4, 18, 20]. That is, we firstly apply a coarse and fast computation. Instead of computing the actual value of $AP(C,F)$ in quadratic time, we may compute a lower bound and an upper bound for $AP(C,F)$ in a constant time $O(1)$. By these bounds, for each cluster C in S, we can rule out the features in a π_j, which are definitely not closest to C; and thus we do not have to precisely compute the average proximity values between these eliminated features and C. Secondly, we can deduce a lower bound and an upper bound for each $PM_{C_0}(C,\Pi)$ from the bounds of AP; and then filter out uninvolved clusters. This is the basic idea of our algorithm. In our algorithm CPM, we have not integrated our algorithm into a particular spatial index, such as R-trees, R^+-trees, etc, due to the following reasons.

- There may be no spatial index built.
- The PM problem may involve many features from different tables/electronic thematic maps; and thus, spatial index built for each thematic map may be different. This brings another difficulty when making use of spatial indices.
- A feature or a cluster, which is qualified in PM, may be only a part of a stored spatial object; for instance, a user may be interested only in certain part of a park. This makes a possible existing index based on the stored spatial objects not quite applicable.

- The paper [18] indicates the existing spatial indexing techniques do not necessarily support well the computation of *aggregate* distances; the argument should be also applied to average distance computation.

The algorithm CPM consists of the following 5 steps:

Step 1: Read the relevant clusters into buffer.

Step 2: Read features batch by batch into buffer and determine their groups by validating the selection conditions against the relational tables.

Step 3: For each feature F in a π_j, compute lower and upper bounds of $AP(C, F)$ for a cluster C. Then determine whether or not F should be kept for the computation of $D(C, \pi_j)$.

Step 4: For each π_j and each cluster C, compute lower and upper bounds for $D(C, \pi_j)$; and then derive lower and upper bounds for (4). Filter out clusters which will not be part of the solution to PM.

Step 5: Apply the above brute-force method to the remaining clusters and their associated features to solve PM.

In the next several subsections we detail the algorithm step by step. Clearly, a success of the algorithm CPM largely relies on how good the lower and upper bounds of AP are. The goodness of lower and upper bounds means two things: 1) the bounds should be reasonably tight, and 2) the corresponding computation should be fast. We first present the lower and upper bounds.

Note that for presentation simplicity, the algorithms presented in the paper are restricted to the case when features and clusters qualified in PM are stored spatial objects in the database. However, they can be immediately extended to cover the case when a feature or a cluster is a part of a stored object.

3.1 Lower and Upper Bounds for Average Proximity

In this subsection, we recall first some useful notation. The *barycenter* (*centroid*) of a cluster C is denoted by $b(C)$. A *convex* [27] polygon encompassing a feature F is called a bounding convex polygon of F. The smallest bounding convex polygon of F is called the *convex hull* [27] of F and is denoted by P_F. An *isothetic* rectangle is orthogonal to the coordinate axis. The minimum bounding rectangle of F refers to the minimum isothetic bounding rectangle of F and is denoted by R_F. Similarly, we denote the convex hull of a cluster C by P_C and denote the minimum bounding rectangle of C by R_C. The bounds presented in the subsection are based on either minimum bounding rectangles or convex hulls.

Given a R_C and a R_F, an immediate idea is to use the shortest distance and the longest distance between R_C and R_F to respectively represent a lower bound and an upper bound of $AP(C, F)$. However, this immediate idea has two problems. The first problem is that when two rectangles intersect with each other (note that in this case C and F do not necessarily have an intersection), the shortest and longest distances between R_C and R_F are not well defined. The second problem is that the bounds may not be very tight even if the two rectangles do not intersect. These also happen similarly for convex hulls. Below, we present new and tighter bounds.

Our lower bound computation is based on the following Lemma.

Lemma 1 $\sum_{i=1}^{K} \sqrt{x_i^2 + y_i^2} \geq \sqrt{(\sum_{i=1}^{K} x_i)^2 + (\sum_{i=1}^{K} y_i)^2}$

Proof: It can be immediately verified that the inequality holds when $K = 2$. By mathematical induction, we can prove the Lemma. □

Theorem 1 *Suppose that C is a cluster, F is a feature, and P is either the convex hull or the minimum bounding rectangle of F. Then, $AP(C,F) \geq d(b(C),P)$; in other words $d(b(C),P)$ is a lower bound of $AP(C,F)$.*

Proof: According to our assumption, every point $p_i \in C$ is outside F. This implies that for each $p_i \in C$, there is a point fp_i in the boundary of F such that $d(p_i, F) = d(p_i, fp_i)$. Suppose that there are N points in C. By Lemma 1, we have

$$AP(C,F) \geq d(b(C), \frac{\sum_{i=1}^{N} fp_i}{n}).$$

Note that $\frac{\sum_{i=1}^{N} fp_i}{n}$ must be contained in the convex hull of F; this means that $\frac{\sum_{i=1}^{N} fp_i}{n}$ is contained in P. Therefore, if $b(C)$ is outside P, then

$$d(b(C), \frac{\sum_{i=1}^{N} fp_i}{n}) \geq d(b(C), P).$$

Note that if $b(C)$ is inside P, then $AP(C,F) \geq 0 = d(b(C),P)$. Thus, the Theorem holds. □

Figure 1 gives an example, and shows that our lower bound is tighter than the shortest distance between two rectangles.

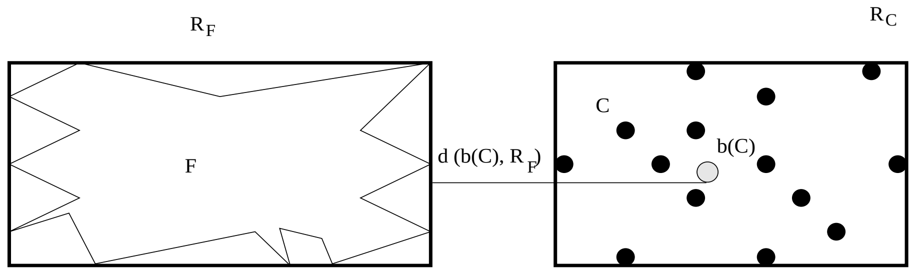


Figure 1: A lower bound

The R_F of a feature F has four edges: the left boundary, right boundary, bottom boundary, and top boundary. Note that each boundary edge x is divided into several line segments (at least two). The two end points of such a line segment are either a) a pair of two adjacent intersection points between F and R_F, or b) a vertex of R_F and an intersection point between F and R_F. We use $HR_{F,x}$ to denote the maximal segment length among these line segments in the boundary edge x, where $x \in \{l, r, b, t\}$ respectively represents either the left, or right, or bottom, or top boundary. For each boundary edge x, we use $VR_{F,x}$ to denote the maximal length of the perpendicular line segment from a vertex of an edge of F, which *faces* [27] x. Figure 2(a) illustrates these concepts.

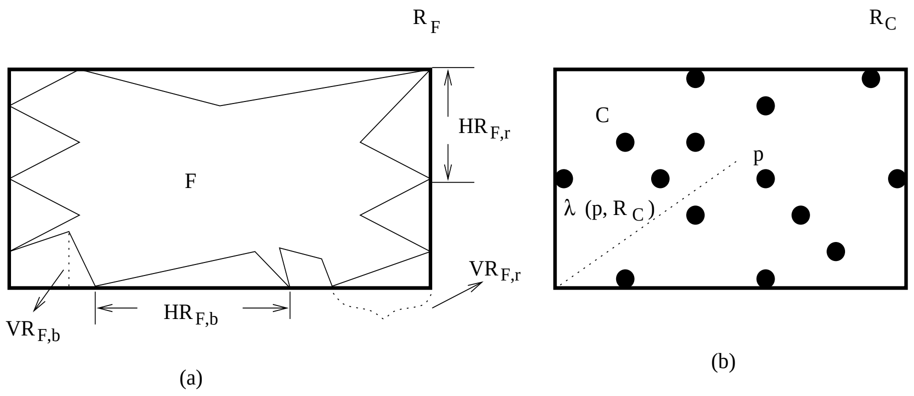


Figure 2: Examples

Suppose that P' is either the convex hull or the minimum bounding rectangle of a cluster C, and p is a point inside P'. We use $\lambda(p, P')$ to denote the maximal distance between p and a point contained in P'. Clearly, $\lambda(p, P')$ is the maximal distance from p to one of the vertices of P'. Figure 2(b) illustrates this concept for the minimum bounding rectangle.

Below we present two upper bounds respectively for convex hulls, and minimum bounding rectangles.

Theorem 2 *Suppose that P_C is the convex hull of a cluster C, p is a point inside P_C, and F is a feature. Then $AP(C, F) \leq d(p, F) + \lambda(p, P_C)$.*

Proof: There two cases 1) p is not outside F, and 2) p is outside F. Recall our assumption in PM that each point in C is outside F. In case 1, we have $d(p', p) \geq d(p', F)$ for each point $p' \in C$ because p is inside F or on the boundary of F. Since $\lambda(p, P_C) \geq d(p', p)$, we have that $\lambda(p, P_C) \geq AP(C, F)$. This proves case 1.

In case 2, we assume that fp is the point on the boundary of F such that $d(p, F) = d(p, fp)$. Note that for each point $p' \in C$, we have:

$$d(p', F) \quad \leq \quad d(p', fp) \leq d(p', p) + d(p, fp) \leq \lambda(p, P_C) + d(p, F)$$

Thus the theorem also holds for case 2. □

It is clear the right hand side in the inequality of Theorem 2 can be used as an upper bound of AP. However, the computation of $d(p, F)$ runs in time $O(|F|)$. The lower bound presented below in Theorem 3 is based on the minimum bounding rectangles and can be computed in constant time, though it is not as tight as that in Theorem 2. First we should note that Theorem 2 also holds if we replace P_C by R_C.

Theorem 3 *Suppose that R_C of C is given, R_F of F is given, and p is a point contained in R_C. Then*

$$d(p, F) + \lambda(p, R_C) \leq \min_{x \in \{l,r,b,u\}} \{\min\{HR_{F,x}, VR_{F,x}\} + d(p, x)\} + \lambda(p, R_C). \tag{5}$$

Here x a boundary edge of R_F.

Proof: From the definition of HR and VR, the theorem can be immediately verified. □

Theorem 3 together with Theorem 2 imply the right hand side in the inequality of Theorem 3 is another upper bound of AP. Further, it should be clear that this upper bound can be computed in constant time, provided that $HR_{F,x}$ and $VR_{F,x}$ are obtained and $\lambda(R_C, p)$ is computed for a given p.

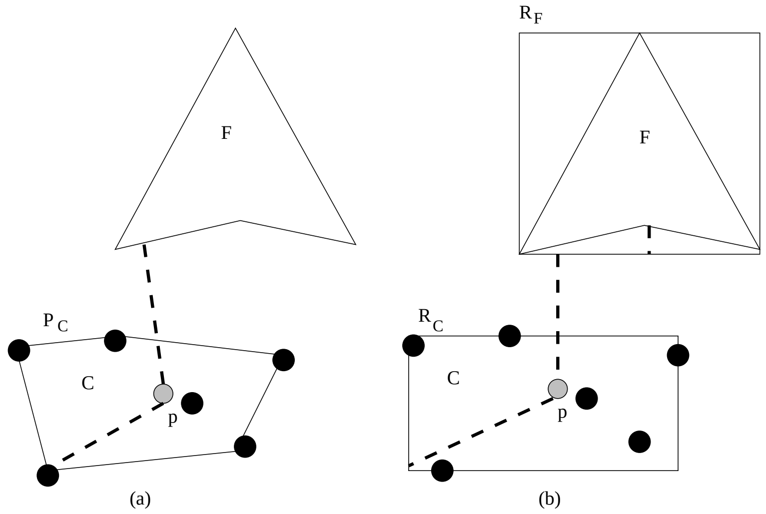


Figure 3: Upper bounds

The total lengths of the thick dotted lines in Figure 3(a) and Figure 3(b) respectively show the upper bounds in Theorems 2 and 3. They also show that our bounds may be tighter than the longest distance between the convex hulls or between the minimum bounding rectangles. However, we cannot generally prove this because the tightness of the bounds depends on the choice of p. In our algorithm, we will choose the *centroid* of a cluster C in the upper bound computation since it has to be used to obtain a lower bound.

In the next several subsections, we present first the algorithm CPM based on the minimum bounding rectangles.

3.2 Read in Clusters

In Step 1, we first read in the clusters, specified in the query by a user, into a buffer by execution of the relational operations [3]; for instance, regarding the query in Example 1 the residential areas with average house price less than $400,000 are read into the database by querying the REAL ESTATE table. Then, we compute R_C, $b(C)$, and $\lambda(b(C), R_C)$ for each cluster C. Clearly, this step takes linear time with respect to the total size of clusters.

3.3 Read in and Filter Out Features

This subsection presents Step 2 and Step 3. Consider that the number of the features to be processed may be very large, and each feature may have many edges. It may be impossible to keep all features in buffer all the time. Consequently, features should be read into buffer batch by batch. Once a batch of features are read in, they are first assigned group IDs against users specifications; for instance, in Example 1 three feature groups are retrieved.

After a feature is assigned a group ID, the algorithm CPM invokes the filtering process in Step 3. It is based on the following lemma.

Lemma 2 *Suppose that C is a cluster, and $\pi = \{F_1, F_2, ..., F_k\}$ is a group of features. For $1 \leq j \leq k$, let $LB_{AP}(C, F_j)$ and $UB_{AP}(C, F_j)$ be a lower bound and an upper bound of $AP(C, F_j)$. Then:*

$$\min_{1 \leq j \leq k} \{LB_{AP}(C, F_j)\} \leq D(C, \pi) \leq \min_{1 \leq j \leq k} \{UB_{AP}(C, F_j)\}.$$

Proof: The lemma immediately follows from the definition of D. □

For notation simplicity, we extend S to include the given C_0; that is, $S = \{C_i : 0 \leq i \leq n\}$. With respect to each pair of C_i and π_j, we use $LB^{i,j}$ to record the minimum value of the lower bounds of $AP(C_i, F)$ for each $F \in \pi_j$, and use $UB^{i,j}$ to record the minimum values of the upper bounds. Lemma 2 says that $LB^{i,j}$ and $UB^{i,j}$ are relatively a lower and an upper bound of $D(C_i, \pi_j)$.

Our algorithm applies an on-line processing paradigm. We initially set both $LB^{i,j}$ and $UB^{i,j}$ to ∞, and then gradually update them when a new feature in π_j is processed. We use a dynamic array $A_{i,j}$ to store the candidate features in π_j for computing $D(C_i, \pi_j)$. $A_{i,j}$ keeps only the features F such taht $LB_{AP}(C_i, F)$ is smaller than the current value of $UB^{i,j}$. Each element in $A_{i,j}$ stores the identifier FID of a feature F, the obtained lower bound of $AP(C_i, F)$, and the pointer *g_des* of F that points to the spatial description of the feature.

Specifically, to process a $F \in \pi_j$, CPM firstly computes R_F and the values of VF and HF for R_F; this can be done easily by scanning F only once. Secondly, for each C_i, we check if F should be included in $A_{i,j}$. According to Lemma 2, we add F to $A_{i,j}$ if and only if the obtained lower bound of $AP(C_i, F)$ is less than the current value of $UB^{i,j}$. Further, we should update $LB^{i,j}$ and $UB^{i,j}$ each time after F is added to $A_{i,j}$; and then check $A_{i,j}$ to determine if some features in $A_{i,j}$ should be removed due to an update of $UB^{i,j}$. To prevent unnecessary scan of $A_{i,j}$ each time after $UB^{i,j}$ is updated, we also record the maximum value of the lower bounds of $AP(C_i, F)$ among the features F in the current $A_{i,j}$; it is recorded by $q^{i,j}$ ($q^{i,j}$ is initially zero); that is, some features in $A_{i,j}$ need to be removed only when $q^{i,j} \geq UB^{i,j}$. Clearly, $q^{i,j}$ needs to be re-computed after certain features are removed from $A_{i,j}$; this is why we keep $LB_{AB}(C_i, F)$ in $A_{i,j}$ for each feature in $A_{i,j}$.

For example, suppose that the features F_1, F_2, F_3, and F_4 are in π_1. Their lower and upper bounds of AP values against a cluster C_1 are $(4, 5)$ for F_1, $(6, 7)$ for F_2, and $(3, 6)$ for F_3, and $(3, 3.5)$ for F_4. Initially, $LB^{1,1} = UB^{1,1} = \infty$, and $q^{1,1} = 0$. Suppose that F_1 is first processed. We add F_1 to $A_{1,1}$ by recording its ID and the lower bound; and then $LB^{1,1} = 4$, $UB^{1,1} = 5$, and $q_{1,1} = 4$. Next, F_2 is processed. F_2 should not be included in $A_{1,1}$ because $6 > UB^{1,1}$. However, when F_3 is processed thirdly, F_3 should be added in $A_{1,1}$. Consequently, $LB^{1,1} = 3$, $UB^{1,1} = 5$, $q^{1,1} = 4$. While processing F_4, we find that F_4 should be added to $A_{1,1}$. Accordingly, $LB^{3,3} = 3$, $UB^{1,1} = 3.5$, and $q^{1,1} = 4$. Then, since $q^{1,1} > UB_{1,1}$ we should check $A_{1,1}$ to delete F_1 from $A_{1,1}$.

More precisely, to process a feature $F \in \pi_j$, the above processes can be presented by the following pseudo codes:

compute R_F, $VF_{R,x}$ and $HF_{R,x}$; // $x = \{l, r, b, t\}$
for each cluster C_i **do** {
  $LB_{AP}(C_i, F) \to a_1$ (the lower bound); $UB_{AP}(C_i, F) \to a_2$ (the upper bound);
  if $a_1 < UB^{i,j}$ **then** {
    $F \to A_{i,j}$;
    $\min\{a_1, LB^{i,j}\} \to LB^{i,j}$;
    $\min\{a_2, UB^{i,j}\} \to UB^{i,j}$;
    $\max\{q^{i,j}, a_1\} \to q^{i,j}$;
    if $q^{i,j} \geq UB^{i,j}$ **then** {
      remove features F from $A_{i,j}$ if $LB_{AP}(C_i, F) \geq UB^{i,j}$;
      re-compute $q^{i,j}$ from the remaining features in $A_{i,j}$; }
  }
}

Once a batch of features are processed by the above procedure, we do not keep them in the buffer if we need to handle several other batches of features. In this situation, we will need to read in again the features not filtered out after Step 4, so that we can process Step 5; this is why we want to keep the pointer *g_des* for each feature object.

3.4 Filter Out Clusters

This subsection describes Step 4. After the computation in the last subsection, we obtained $LB^{i,j}$ and $UB^{i,j}$ for each pair of C_i and π_j. By Lemma 2, we have:

$$LB^{i,j} \leq D(C_i, \pi_j) \leq UB^{i,j}. \tag{6}$$

Note that from the definition, $pm(D(C_i, \pi_j), D(C_0, \pi_j)) = 0$ if $D(C_i, \pi_j) \leq D(C_0, \pi_j)$. This immediately implies that $pm(D(C_i, \pi_j), D(C_0, \pi_j)) = 0$ if one of the following two conditions applies:

1. $UB^{i,j} \leq LB^{0,j}$, or
2. $UB^{i,j} \leq D(C_0, \pi_j)$

Thus, intuitively $pm(D(C_i, \pi_j), D(C_0, \pi_j))$ is bounded by $UB^{i,j} - LB^{0,j}$. Further, it cannot be smaller than the minimum distance between the two closed intervals: $[LB^{i,j}, UB^{i,j}]$ and $[LB^{0,j}, UB^{0,j}]$. The intuition can be immediately verified and is stated in Lemma 3. Let

$$P^{i,j} = \begin{cases} UB^{i,j} - LB^{0,j} & \text{if } UB^{i,j} > LB^{0,j} \\ 0 & \text{otherwise} \end{cases}$$

and let

$$\alpha^{i,j} = \begin{cases} LB^{i,j} - UB^{0,j} & \text{If } LB^{i,j} > UB^{0,j} \\ 0 & \text{otherwise} \end{cases}$$

Lemma 3 *$pm(D(C, \pi_j), D(c_i, \pi_j)) \leq P^{i,j}$, and $pm(D(C, \pi_j), D(c_i, \pi_j)) \geq \alpha^{i,j}$ for each pair of C_i and π_j,*

Proof: From (6), it follows that:

$$LB^{i,j} - UB^{0,j} \leq D(C_i, \pi_j) - D(C_0, \pi_j) \leq UB^{i,j} - LB^{0,j}. \tag{7}$$

This together with the above two conditions derive the first inequality. The second inequality immediately follows from the definitions of $a^{i,j}$ and pm. $\Box$

Lemma 3 gives us a lower and upper bound of $PM_{C_0}(C_i, \Pi)$ for each C_i:

$$\sum_{j=1}^{m} w_j \alpha^{i,j} \leq PM_{C_0}(C_i, \Pi) \leq \sum_{j=1}^{m} w_j P^{i,j}. \tag{8}$$

In Step 4, we firstly compute the lower and upper bounds of $PM_{C_0}(C_i, \Pi)$ for each cluster C_i ($i \neq 0$), as given in (8). Secondly, we compute the minimum value τ of the upper bounds among the clusters C_i in S but $i \neq 0$. Thirdly, we scan S to filter out clusters C_i if $\sum_{j=1}^{m} w_j \alpha^{i,j} > \tau$. This procedure runs in time $O(nm)$, where n is the number of clusters and m is the number of groups of features.

3.5 Precise Computation

This subsection describes Step 5. After pruning the clusters, the information of remaining features for solving PM is kept in each $A_{i,j}$. We need to read in these features again to perform the precise computation. Since two different $A_{i,j}$s may keep a same feature as a candidate, to efficiently read in the required features we use a hashing method. A hash table H is created to store the remained features against feature ID - FID. We scan $A_{i,j}$ one by one to execute the following two steps.

Step 1: For each feature $F \in A_{i,j}$, hash its FID into an H entry and then determine if its spatial description is already in H.

Step 2: If the spatial description of F is not in H, then read it into buffer using g_des and store it in H. Then, compute $AP(C_i, F)$ for each remaining C_i.

After computing all AP values, we apply the remaining step of the brute-force method to the existing clusters to solve the problem. Note that before starting Step 5, we also check a special case - if $\alpha^{i,j} = P^{i,j}$, for all remaining clusters and features; then CPM does not have to process Step 5 but outputs the remaining clusters as the solution.

3.6 Implementation and Complexity of CPM

In this subsection, we outline our actual implementation of CPM, analyze the complexity of CPM.

In Step 1, we use a linked list to read in clusters in the buffer as well as to compute and store the information of R_C, $b(C)$, and $\lambda(b(C), R_C)$ for each cluster C. Clearly, Step 1 takes linear time with respect to the total sizes of clusters; that is, $O(\sum_{i=0}^{n} |C_i|)$.

In Step 2, we use another linked list to read in features batch by batch. We implement this in a simple way - each time we read into buffer a fixed number of features, which is determined by the buffer size and the average size of the features. Step 2 takes time $O(\sum_{F\in\sum_{j=1}^{n}\pi_j}|F|)$, where n is the number of features.

Step 3 is combined with Step 2 with respect to each batch of data read in. With respect to each pair of cluster C_i and a $\pi_j \in \Pi$, a linked list $A_{i,j}$ is created to store the information of $LB^{i,j}$, and $UB^{i,j}$, as well as IDs of the features in π_j to be remained with respect to C_i and Π_j. Note that in the linked list, we do not record the details of a feature to be remained, rather we record its ID and the physical address of the geometric data. According to the algorithm described in section 3.4, Step 3 runs in time $O(\sum_{F\in\sum_{j=1}^{n}\pi_j}|F| + n\times\sum_{j=1}^{m}|\pi_j|)$, where m is the number of the elements in Π.

Step 4 takes time $O(nm)$, while step 5 takes time $O(\sum_{\forall C_i,\forall F\in A_{i,j}}|C_i||F|)$ for the remaining clusters C_i.

Note that the complexity of the brute-force algorithm runs in time $O(\sum_{\forall C,\forall F}|C||F|)$. It should be clear that the time complexity of the brute-force method is much higher than that of CPM, when

- the average size of each cluster is large, and
- the average size of each feature is large, and
- the algorithm significantly filters out the clusters.

This is confirmed by our experiment results in Section 4.

3.7 Different Modes of CPM

The above mode of CPM uses only the minimum bounding rectangles; and it is denoted by CPM-R.

An alternative mode to CPM-R is to use a multiple-filtering technique [2, 13, 18]:

- first the minimum bounding rectangles are used in Steps 3 and 4, and then
- the convex hulls for features and clusters are adopted to repeat Steps 3 and 4 before processing Step 5.

It is denoted by CPM-RH. In CPM-RH, we employ the divide and conquer algorithm [27] to compute convex hulls for clusters. To compute a convex hull for a feature (simple polygon), we employ the last step in Graham's scan [27], which runs in linear time. We use the upper bound of AP in Theorem 2 to implement the procedure in Section 3.3 for convex hulls.

Another alternative mode to CPM-R is to use only the convex hulls instead of the minimum bounding rectangles. We denote this mode by CPM-H.

In next section, we will report our experiment results regarding the performances of the brute-force algorithm, CPM-R, CPM-RH, and CPM-H.

4 Experiment Results

The brute-force algorithm and the three different modes of CPM have been implemented in C++ on a Pentium I/200 with 128 MB of main memory, running Window-NT 4.0. In our experiments, we evaluated the algorithms for efficiency and scalability. Our performance evaluation is basically focused on Step 3 onwards, because the methods [3] of reading in clusters and features are not our contribution. Therefore, in our experiment we record only the CPU time but exclude I/O costs.

In our experiment, we developed a program to generate a benchmark. This is partly because we do not have a real data at hand, and partly because we want evaluate a "general" performance of the algorithm. In the program, we first use the following random parameters to generate rectangles, such that a rectangle may intersect with at most another rectangle:

- M gives the number of rectangles, and
- wid_R controls the width of a rectangle R and h_R controls the height of R.

More specifically, we first generate M rectangles R with a random width wid_R and a random height h_R, where $1 \leq wid_R, h_R \leq 1000$. The generated rectangles are randomly distributed in the 2-dimensional space, and intersect with at most another rectangle. To generate n clusters, we randomly divide the whole region into n disjoint sub-regions and choose one rectangle from each region. We use a random parameter NC to control the average number of generated points in each rectangle among the chosen n rectangles. These give n clusters. The remaining $M - n$ rectangles correspond to features. We use another random parameter NF to control the average number of the vertices (points) generated in each remaining rectangle.

Note that if R intersects with R', we actually generate the points respectively in R and $R' - R$ if R' is not included in R. Further, in each rectangle R corresponding to a feature, we apply a Graham's scan-like algorithm [27] to produce a simple polygon connecting each vertex in R; the generated simple polygon is used as a feature. Therefore, in our benchmark we have two kinds of spatial objects - clusters and features.

In the experiments below, we adopt a common set of parameters: 1) a feature has 150 edges on average, 2) a cluster has 300 points on average, and 3) the features are grouped into 20 groups.

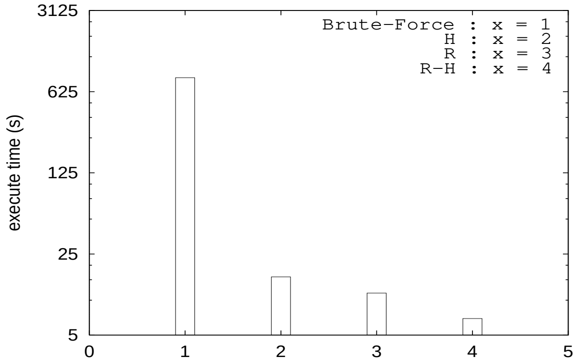
In our first experiment, we generate a database with 10 clusters and 1000 features. The experiment results are depicted in Figure 4, where the algorithm CPM-R, CPM-H, and CPM-RH are respective abbreviated to "R", "H", and "R-H".

From the first experiment, we can conclude that the brute-force is practically very slow. Another appearance is that H is slower than R and R-H due to the fact that in H, the computation of lower and upper bounds for each pair of cluster and feature does not take constant time. Intuitively, H should be significantly slower than R and R-H when the number of clusters and features increases; this has been confirmed by the second experiment.

The second experiment has been undertaken through two "dimensions":

- Fix the number of features to be 1000, while the number of clusters varies from 5 to 20. The results are depicted in Figure 5.
- Fix the number of clusters to be 200, while the number of features varies from 1000 to 10000. The results are depicted in Figure 6.

Note that in the second experiment, we run R, H, and $R - H$ 40 times against each database. This experiment, together with the first experiment, also demonstrates that $R - H$ is faster than H on average.

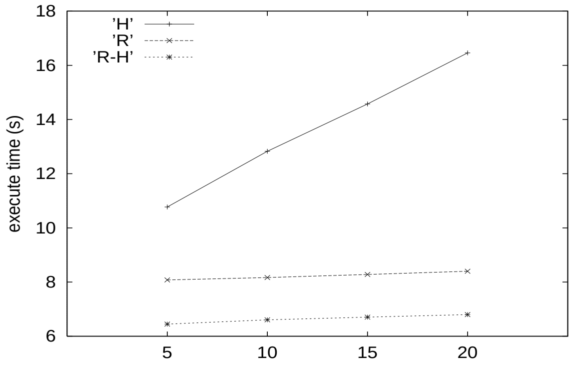


(a) histogram presentation

Algorithm	Average Run Time (s)
Brute-force	823.514
H	15.863
R	11.516
R-H	6.95

(b) table presentation

Figure 4: A Comparison among four algorithms

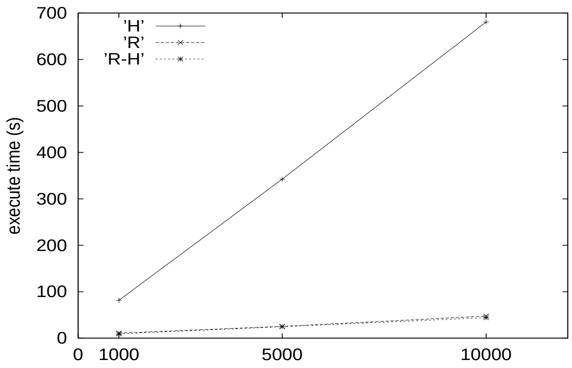


(a) graph presentation

#CL	Algorithm	Average Run Time (s)
5	H	10.772
	R	8.079
	R-H	6.447
10	H	12.827
	R	8.168
	R-H	6.606
15	H	14.575
	R	8.28
	R-H	6.706
20	H	16.458
	R	8.399
	R-H	6.8

(b) table presentation

Figure 5: A Comparison among H, R, and R-H



(a) graph presentation

#FE	Algorithm	Average Run Time (s)
1000	H	81.738
	R	10.929
	R-H	9.267
5000	H	342.38
	R	25.295
	R-H	24.694
10000	H	680.751
	R	47.578
	R-H	44.246

(b) table presentation

Figure 6: Another Comparison among H, R, and R-H

In the third experiment, to test the scalability we vary the database size from 1000 features to 50000 features but fix the the number of clusters to be 200. For each database, we run both R and R-H 40 times against each database. Figure 7 illustrates our experiment results.

The three conducted experiments suggest that our algorithm is efficient and scalable. Secondly, we can see that though an application of convex hulls to our filtering procedures is more accurate than an application of minimum rectangles, but it is too expensive to use directly. The best use of convex hulls should follow an application of minimum bounding rectangles.

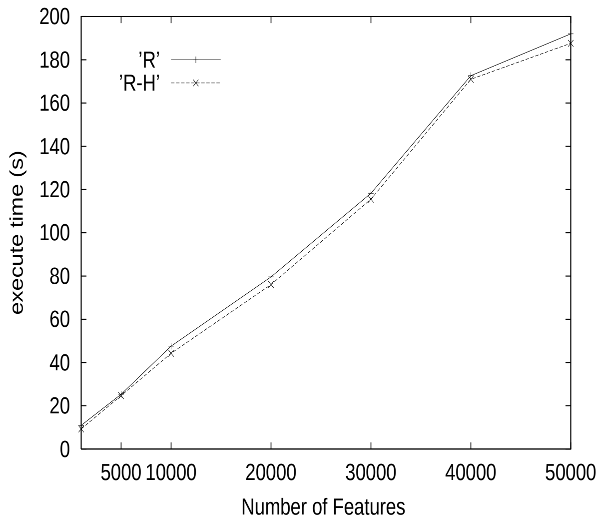
5 Discussion

The problem PM and the algorithm CPM may be either generalized or constrained according to various applications. In this section, we present a discussion on these issues.

A slight modification of the algorithm CPM can be applied to the case where we specify the proximity matching conditions by the shortest distances instead of by specifying a given cluster. For instance, in Example 1 we may directly specify that such areas are within 5 km away from a private school, 6 km away from a beach, and 2 km away from a golf course, instead of comparing to the SeaBreez Real-estate.

Our results and discussions, so far, are limited to the Euclidean distance for computing AP values. Note that the deduction of the upper bounds presented in Section 3.1 are based on a triangular inequality in the Euclidean distance; that is $d(p_1, p_2) \leq d(p_1, p_3) + d(p_3, p_2)$. Since the triangular inequality is part of the definition of any metric distance, a similar form of upper bounds can be applied to any metric space. We should also note that similar lower bounds as those presented in Section 3.1 can be obtained in such a metric space that the metric distance γ follows the two constraints below:

- $\gamma(p_1, p_2) + \gamma(p_3, p_4) \geq \gamma(p_1 + p_3, p_2 + p_4)$, and
- $\gamma(c \times p_1, c \times p_2) = |c|\gamma(p_1, p_2)$ for any constant c.



(a) graph presentation

#FE	Algorithm	Average Run Time (s)
1000	R	10.929
	R-H	9.267
5000	R	25.295
	R-H	24.694
10000	R	47.578
	R-H	44.246
20000	R	79.598
	R-H	76.023
30000	R	118.265
	R-H	115.557
40000	R	172.628
	R-H	171.044
50000	R	191.978
	R-H	187.619

(b) table presentation

Figure 7: A comparison between R and R-H

This is because that our lower bound deducation is based on the above two conditions. Consequently, we can extend the problem PM and the algorithm CPM to any metric space (for computing AP values) where the above two constraints are satisfied. For instance, we can verify that the *Manhattan distance* [27] satisfied the two constraints; and thus our algorithm can be extended to 2-dimensional Manhattan distance space.

6 Conclusions

In this paper, we formalized a new problem (PM) in spatial data mining from real applications. We presented an efficient algorithm based on several novel pruning conditions, as well as various different modes of the algorithm. Our experiment results showed that the algorithm is very efficient and can support a number of real applications where data with huge volume are present.

Further, in Section 5, we showed that our work in this paper can be extended to many other metric spaces.

Note that the PM problem and the algorithm CPM are restricted to the case where a cluster is outside a feature. This restriction may be not generally applicable to some applications; we are now identifying such applications. Besides, we are currently working on the development of indexing techniques to support CPM. Further, a modification of PM to cover the applications where the distance from a cluster to a set of features is not necessarily restricted to one feature in the set seems more complicated; this is our another future study.

References

[1] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules, *Proceedings of the 20th VLDB Conference*, 487-499, 1994.

[2] M. Ankerst, B. Braunmuller, H.-P. Kriegel, T. Seidl, Improving Adaptable Similarity Query Processing by Using Approximations, *Proceedings of the 24th VLDB Conference*, 206-217, 1998.

[3] W. G. Aref and H. Samet, Optimization Strategies for Spatial Query Processing, *Proceedings of the 17th VLDB Conference*, 81-90, 1991.

[4] T. Brinkhoff, H. P. Kriegel, and R. Schneider, and B. Seeger, Multistep processing of spatial joins, *Proc. of ACM SIGMOD*, pp. 197-208, 1994.

[5] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, A density-based algorithm for discovering clusters in large spatial databases, *Proceedings of the Second International Conference on Data Mining KDD-96*, 226-231, 1996.

[6] M. Este, H.-P. Kriegel, J. Sander, Spatial Data Mining: A Database Approach, *SSD'97*, LNCS 1262, 47-65, 1997.

[7] V. Estivill-Castro and A.T. Murray, Discovering Associations in Spatial Data - An Efficient Medoid Based Approach, *Proceedings of the Second Pacific-Asia Conference on Knowledge Discovery*, LNAI 394, 110-121, 1998.

[8] U. M. Fayyad, S. G. Djorgovski, and N. Weir, Automating the analysis and cataloging of sky surveys, *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, 1996.

[9] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, Menlo Park, CA, 1996.

[10] D. Fisher, Improving Inference through Conceptual Clustering, *Proceedings of 1987 AAAI Conferences*, 461-465, 1987.

[11] R. H. Guting, An Introduction to Spatial Database Systems, *VLDB Journal*, 3(4), 357-400, 1994.

[12] R. Guttman, A Dynamic Index Structure for Spatial Searching, *ACM-SIGMOD International Conference on Management of Data*, 47-57, 1984.

[13] J. Han, Spatial Data Mining and Spatial Data Warehousing, *Invited Talk at SSD'97*, 1997.

[14] J. Han, Y. Cai, and N. Cercone, Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases, *IEEE Trans. knowledge and Data Engineering*, 5, 29-40, 1993.

[15] J. Han, K. Koperski, and N. Stefanovic, GeoMiner: A System Prototype for Spatial Data Mining, *Proceedings of 1997 ACM-SIGMOD International Conference on Management*, 553-556, 1997.

[16] G. R. Hjaltason and H. Samet, Incremental Distance Join Algorithms for Spatial Databases, 237-248, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, 1998.

[17] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley & Sons, 1990.

[18] E. M. Knorr and R. T. Ng, Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining, *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 884-897, 1996.

[19] E. M. Knorr, R. T. Ng, and D. L. Shilvock, Finding Boundary Shape Matching Relationships in Spatial Data, *SSD'97*, LNCS 1262, 29-46, 1997.

[20] K. Koperski and J. Han, Discovery of Spatial Association Rules in Geographic Information Databases, *Advances in Spatial Databases*, Proceeding of 4th Symposium (SSD'95), 47-66, 1995.

[21] K. Koperski, J. Han, and J. Adhikary, Mining Knowledge in Geographic Data, to appear in *Communications of ACM.*

[22] R. S. Michalski, J. M. Carbonnel, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufman, 1983.

[23] W. Lu, J. Han, and B. C. Ooi, Knowledge Discovery in Large Spatial Databases, *Proceedings of Far East Workshop on Geographic Information Systems*, 275-289, 1993.

[24] H. Lu, R. Setiono, and H. Liu, NeuroRule: A Connectionist Approach to Data Mining, *VLDB'95*, 251-262, 1995.

[25] N. Ng and J. Han, Efficient and Effective Clustering Method for Spatial Data Mining, *Proceeding of 1994 VLDB*, 144-155, 1994.

[26] J. S. Park, M.-S. Chen, and P. S. Yu, An Effective Hash-Based Algorithm for Mining Association Rules, *Proceedings of 1995 ACM SIGMOD*, 175-186, 1995.

[27] F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[28] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.

[29] G. Shaw and D. Wheeler, *Statistical Techniques in Geographical Analysis*, London, David Fulton, 1994.

[30] H. Toivonen, Sampling Large Databases for Association Rules, *Proceedings of 22nd VLDB Conference*, 1996.

[31] X. Xu, M. Ester, H.-P. Kriegel, Jorg Sander, A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases, *ICDE'98*, 324-331, 1998.

[32] T. Zhang, R. Ramakrishnan and M. Livny, BIRCH: an efficient data clustering method for very large databases, *Proceeding of 1996 ACM-SIGMOD International Conference of Management of Data*, 103-114, 1996.