March 7, 2001
RT0404
Computer Science 20 pages

Research Report

Mining Optimized Distance and / or Orientation Rules in Spatial Databases

Yasuhiko Morimoto, Harunobu Kubo, Takeshi Kanda

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Mining Optimized Distance and / or Orientation Rules in Spatial Databases

Yasuhiko Morimoto
IBM Tokyo Research Laboratory
morimoto@trl.ibm.co.jp

Harunobu Kubo
IBM Tokyo Research Laboratory
kuboh@jp.ibm.com

Takeshi Kanda
University of Tokyo
kanda@simplex.t.u-tokyo.ac.jp

Yasuhiko Morimoto is the contact author.

IBM Tokyo Research Laboratory
1623-14, Shimo-tsuruma, Yamato, Kanagawa 242-8502, Japan

Email: morimoto@trl.ibm.co.jp
Phone: +81-46-273-4918
Fax: +81-46-273-7413

Abstract

We consider the problem of spatial data mining where we find spatial association rules that contain spatial predicates. Various kinds of spatial predicates can be defined in the spatial association rules such as "topological relationship," "distance," "orientation," and so forth. Among them, "distance" and "orientation" are quantitative predicates. In order to use such quantitative predicates in a rule, we have to specify values for them. For example, we set 10 miles distance as a short driving distance. Then, we can find rules that are derived from or that lead to the distance. However, implications of such specific values are differ depending on application domains. Therefore, in a data mining process, we need a fast algorithm for finding objective values for such quantitative predicates. In this paper, we present a data mining system, which uses efficient algorithms in computational geometry, for finding the optimized distance and / or orientation according to a specified criterion.

1 Introduction

Recent progress in computing facilities makes it possible to integrate geographic information system (GIS) and huge databases that contain spatial information such as addresses. Efficient

data management and retrieval in such integrated GIS have been investigated so far [9]. Among them, there are several researches that focused on *spatial data mining*, i.e., mining knowledge from huge spatial databases [16, 12, 4, 11, 10, 20, 3]. With the growth of mobile computing environments, it will be expected that we have huge spatial databases more easily. Therefore, spatial data mining will become much more important.

In data mining literature, association rules [1] are one of the most fundamental knowledge that can be found in (transaction-based) relational databases. Koperski and Han extended the association rules to spatial databases and defined *spatial association rules* [12] that contain *spatial predicates*. Various kinds of spatial predicates can be defined in the spatial association rules such as "topological relationship," "distance," "orientation," and so forth.

Among those spatial predicates, "distance" and "orientation" are quantitative predicates. In order to use such quantitative predicates in a rule, we have to specify values for them. For example, we set 10 miles distance as short driving distance. Then, we can find rules that are derived from or that lead to the distance. However, implications of such specific values are differ depending on application domains. Therefore, in a data mining process, we need a fast algorithm for finding objective values for such quantitative predicates. In this paper, we present a data mining system, which uses efficient algorithms in computational geometry, for finding the optimal distance and / or orientation according to a specified criterion.

Motivating Example

One of the most fundamental spatial searches in a spatial database is a selection facilities based on distance. Assume that we have a spatial database that has tables containing information about customers and stores with the following schemata:

```
customers=(id,name,address,income,occupation)
stores=(id,name,address,revenue)
```

Following query is one of such distance queries.

"Find the name and address of customers who live within 10 miles of a store."

Current advanced spatial databases can answer such query efficiently by using spatial indexes. Conventional spatial data mining systems can apply the technique and finds several spatial association rules concerning the distance. For example, we can find following rules and insight concerning the distance:

- `distance(customers.address, ∃stores.address) < 10`
  `⇒ customers.occupation = 'student'`
  (Many customers **living within 10 miles** of a store are **students**.)

- `customers.income > $10M`
  `⇒ distance(customers.address, ∃stores.address) < 10`
  (Most of customers whose **income is more than $10M** are **living within 10 miles** of a store.)

In this example, we set 10 miles distance before the mining process. The distance may imply a short driving distance in an application domain. After defining the specific distance, we can find rules that are derived from or that lead to the distance.

Main Results

In this paper, we present a data mining system for finding the optimal distance and / or orientation according to a specified criterion. We call association rules that derived from the optimal distance *optimized distance rules*. In the above example, our mining system finds the optimal distance that maximizes the probability of student, if the specified criterion is the maximization of confidence to be student with a minimum support value. Similarly, it can find the optimal distance that maximizes the probability of customers whose income is more than $10M, if the specified criterion is the maximization of the probability of such wealthy customers. And the system finally outputs optimized distance rules by using the optimal distances as follows:

- `distance(customers.address, ∃stores.address) < 2`
  `⇒ customers.occupation = 'student'`

(Many customers **living within 2 miles** of a store are **student**. And, the **2 miles** distance maximizes the confidence, i.e., the probability of **student** customers in customers who live within the distance, subject to a given minimum support, i.e., the probability of customers who live within the distance in all customers in the database.)

- `customers.income > $10M`

  $\Rightarrow$ `distance(customers.address, ∃stores.address) < 12`

  (Most of customers whose **income is more than $10M** are **living within 12 miles** of a store. And, the **12 miles** distance maximizes the confidence, i.e., the probability of customers whose **income is more than $10M** in customers who live within the distance, subject to a given minimum support.)

Notice that criteria, "maximization of the probabilities of specific kinds of customers," used in the example are subjective features. If the user do not have a specific intention for such criteria, the system can extract optimal distance rules by using general criteria such as "gain of mutual information," "confidence" (for categorical non-spatial attribute) and "mean squared error" (for numerical non-spatial attribute).

As for the "orientation" predicate, conventional spatial data mining systems have to define a specific range of angles before a mining process. For example, we define "`east_of`" predicate as the angle range from 45 degrees to 135 degrees. Then, a mining system tries to find association rules concerning the angle range. We can similarly define *optimized orientation rules* as well for "orientation" predicates. For the optimized orientation rules, we compute the optimal angle ranges according to a criterion.

Figure 1 shows an example of the optimal distance rule and orientation rules on a map.

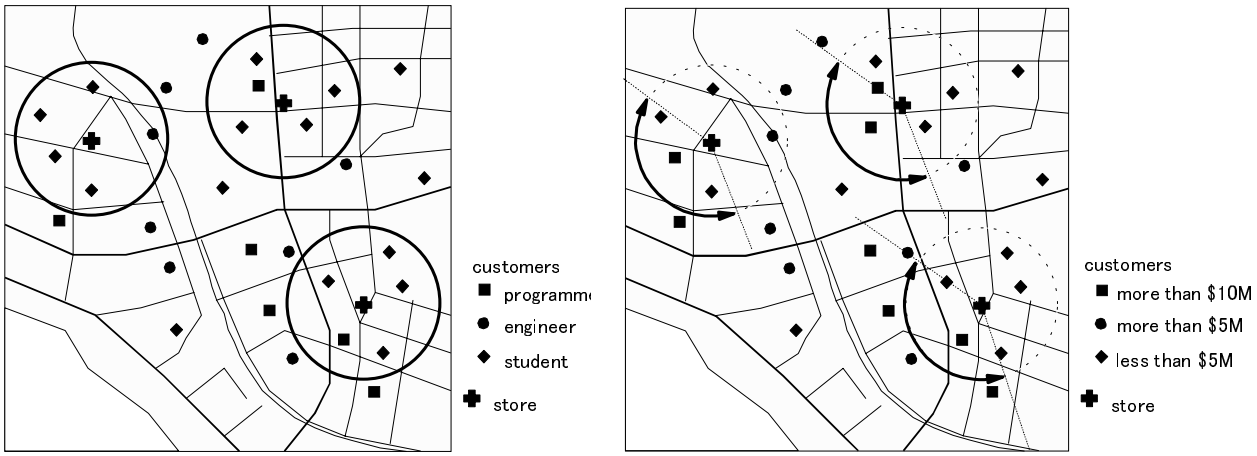

Figure 1: Optimized Distance Rule (left) and Optimized Orientation Rule (right)

2 Preliminaries

Spatial Association Rules

In the definition of Koperski and Han [12], spatial association rules have a form "$X \Rightarrow Y$ (c)" where X and Y are sets of spatial and non-spatial predicates and c is confidence of the rule. It means that when X occurred, Y also occurred with the probability of c. In their definition, a spatial association rule contains at least one spatial predicate in the form.

A spatial predicate is one of the following relations between a spatial object and another spatial object (or other spatial objects):

- topological relationship={overlap, disjoint, contain, etc.}
- distance={close_to, far_from, etc.}
- orientation={north_of, west_of, south_of, east_of, etc.}

Each spatial object can be evaluated whether the object satisfy a predicate of these or not.

Among those predicates, distance and orientation are substantially quantitative predicates. In order to evaluate whether an object satisfies such quantitative predicates or not, we have to predefine the specific quantity for each predicate. The "close_to" predicate, for

example, we may define the predicate as Euclidean distance between two objects is less than 5 miles. However, adequate definitions for such quantitative predicates are different for each application domain. Moreover, important hidden knowledge might be in distance or orientation between spatial objects. Therefore, it is highly demanding to compute objective and significant values for such quantitative predicates.

Example of Spatial Objects

In the rest of the paper, we assume a spatial database that has tables with following schemata:

customers=(id,name,address,income(N),occupation(C))

occupation={student,programmer,engineer}

police offices=(id,name,address)

post offices=(id,name,address)

schools=(id,name,address,category(C))

category={elementary,junior high,high}

stations=(id,name,address)

stores=(id,name,address,revenue(N))

In the schemata, id, name, and address are common features of spatial objects. The underlined attributes are additional non-spatial features and (N) indicates a numerical feature while (C) indicates a categorical feature. The categorical attribute occupation of **customers** table can take 'student,' 'programmer,' or 'engineer' as its value. The categorical attribute category of **schools** table can take 'elementary,' 'junior,' or 'high.'

Starting Points and Query Points

We define sets of *starting points* and sets of *query points* to compute distance or orientation. We consider distance and orientation between a point in starting points and a point in query points.

We can choose several kinds of spatial objects from databases and define sets of each kind as sets of starting points. For example, we may define seven sets of staring points by chosing "police offices," "post offices," "elementary schools," "junior high schools," "high schools," "stations," and "stores."

Next, we define *query points* that are used as population sets of all optimized distance / orientation rules. Each set of query points must have non-spatial features (attributes) to compute a value of the objective function such as "information gain," "mean squared error," and so forth. For example, we may define query points as "customers" and "stores."

Distance for each query point is Euclidean distance between the query point and the nearest starting point from the query point. Similarly, orientation for each query point is an angle from the nearest starting point to the query point. The distance can be defined for each set of starting points.

Ordered Distance / Orientation Tables

After defining sets of starting points and sets of query points, we can compute distance and orientation. In order to simplify the explanation, we focus on finding optimized distance rules. Remind that we can compute optimized orientation rules as well by using the same procedure.

First of all, the mining system creates intermediate relational tables that contain necessary information to compute the optimal distance. The intermediate tables consist of (summarized or bucketed) records that is ordered by distance value. If the number of query points, say n, in a set is small enough to sort and store all information that is necessary to evaluate a specified criterion, we can compute accurate optimal distance without a bucketing with distance value. However, the number of query points tends to be large in a huge spatial database and the sorting operation, i.e., $O(n \log n)$, and the space for intermediate tables might be costly. Moreover, we have to create similar ordered tables for all combinations of query point sets and starting point sets. Therefore, the mining system creates ordered buckets that have the same size of user specified distance interval so that the number of buckets (records in an intermediate table), say B, becomes $B \ll n$.

Table 1: Ordered Distance Table ("Customers" and "Police Offices")

bkt index	distance	#records	$\sum$ income	#programmer	#engineer	#student
(1)	0...1	28	921.53	7	10	11
(2)	1...2	31	1021.41	10	11	10
(3)	2...3	21	1106.03	7	9	5
⋮	⋮	⋮	⋮	⋮	⋮	⋮
(B)	20...	8	29.50	3	4	1

Table 1 shows an example of such intermediate relational tables. Each bucket (record) has one mile interval of distance and contains summerized information of query points that belong to the corresponding disance range.

Optimized Distance / Orientation Rules

Once we create the ordered distance tables, the problem to compute an optimized distance rule is the same as finding a cutting point (or range for orientation rules) in the tables so that the cutting point optimizes one of following criteria:

- For Non-Spatial Categorical Attribute:
  - Gain of mutual information (entropy) or GINI index or Chi square value based on value distribution of the attribute
  - Maximization of confidence, of each attribute value, subject to minimum support
  - Maximization of support subject to minimum confidence of each attribute value
- For Non-Spatial Numerical Attribute:
  - Minimization of mean squared error (deviation) of attribute value

Detailed definitions of those criteria are defined in [7, 15, 14, 13].

Since the number of records of each ordered table is B ($B \ll n$), we can find such optimal cutting point in $O(B)$ time. However, for orientation rules, we have to compute optimal range.

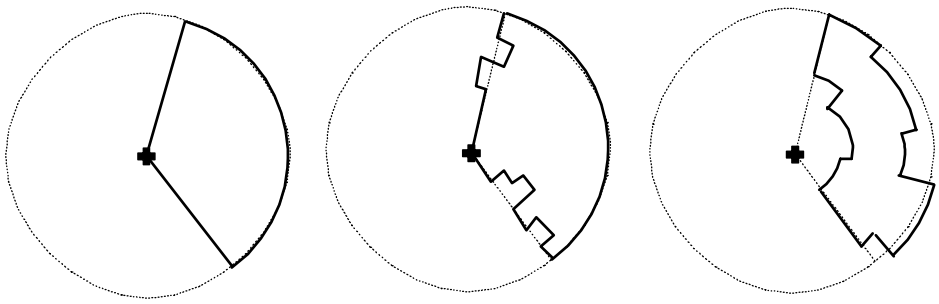

Figure 2: Two-Dimensional Rules of Distance and Orientation

In such cases, there is an $O(B)$ time algorithm to find optimal range for the maximization of confidence or support [7, 6]. As for the other criteria, there is an efficient algorithm that is expected to run in $O(B \log B)$ time [15, 14].

We can also consider distance and orientation simultaneously. If there are two numeric attribute for predicate, in this case "distance" and "orientation," we can compute the optimized two-dimensional rule efficiently [5, 15, 14]. We can use rectangular, x-monotone, and rectilinear regions as the optimized two-dimensional rules. Figure 2 are examples of a rectangular (left) rule and an x-monotone (middle,right) rules on maps.

If we compute the ordered distance tables, we can use such efficient algorithms to compute the optimized distance and / or orientation rules. Therefore, in this paper, we focus on efficient constructions of the ordered distance tables in the next section.

3 Indexing Structures and Algorithms

Finding the Nearest Starting Point

For each query point, we have to find the nearest starting point and compute distance and orientation. One of an efficient data structure for the purpose is, what is called, Voronoi diagram [2]. We construct Voronoi diagrams for each set of starting points.

Let a set of n query points be $Q = \{q_1, ..., q_n\}$, and let a set of m starting points be $S = \{s_1, ..., s_m\}$. And, let $distance(q, s)$ be the Euclidean distance between two points q and

s in the two-dimensional plane. The Voronoi diagram of S is the subdivision of the plane into m regions, say "Voronoi regions," one for each point in S. Each Voronoi region has the property that a point q lies in the region corresponding to a starting point s_i if and only if $distance(q, s_i) < distance(q, s_j)$ for each $s_j \in S$ with $j \neq i$. We denote the Voronoi diagram of S by $Vor(S)$ and denote a Voronoi region that corresponds to a starting point s_i by $Reg(s_i)$ in this paper.

Algorithm 3.1 finds the nearest starting point $s_{nearest} \in S$ for a query point $q \in Q$ by using the Voronoi diagram $Vor(S)$. Though the worst time complexity of Algorithm 3.1 is $O(m)$, the expected running time becomes constant if we can start the algorithm from a point that lies close to the nearest point. A method presented in [17, 18] uses a quaternary tree bucketing and finds the nearest point efficiently.

Algorithm 3.1 Point Location of Voronoi Diagram [1]

Input: Point (q), Voronoi diagram ($Vor(S)$) **Output:** Point ($s_{nearest}$)

(1) Choose a starting point $s_i \in S$ arbitrary.

(2) For each s_n whose $Reg(s_n)$ is adjacent to $Reg(s_i)$:

  (a) If $distance(s_n, q) < distance(s_i, q)$, set s_i be s_n and go to (2).

(3) Answer s_i as $s_{nearest}$.

Constructing Voronoi Diagram

We need Voronoi diagrams for each set of starting points. We use an algorithm, called "incremental method" [19, 8], to construct Voronoi diagrams. Assume there is a Voronoi diagram $Vor_i(S_i)$ of a set S_i with i points. An incremental method adds new point s_{i+1} into the $Vor_i(S_i)$ and constructs $Vor_{i+1}(S_{i+1})$ like in Figure 3. Algorithm 3.2 is one of such methods.

Algorithm 3.2 Incremental Construction of Voronoi Diagram [1]

[1] In order to simplify the explanation, we omitted some exceptional conditions, for example, a condition when q lies on the border of Voronoi regions.

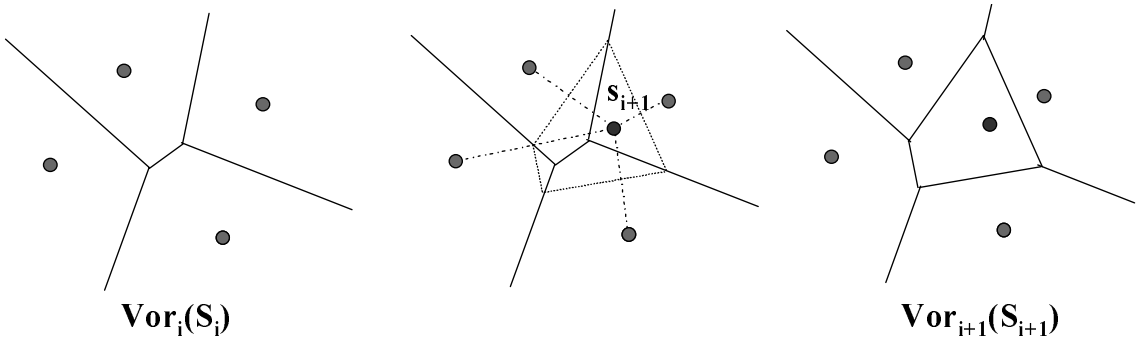

Figure 3: Incremental Update of Voronoi Diagram

Input: Point (s_{i+1}), Voronoi diagram ($Vor_i(S_i)$) **Output:** Voronoi diagram ($Vor_{i+1}(S_{i+1})$)

(1) Initialize set of segments B to be an empty set.

(2) Find the nearest point $s \in S_i$ by using Algorithm 3.1 with parameters s_{i+1} and $Vor_i(S_i)$.

(3) Divide $Reg(s)$ with the perpendicular bisector of s and q.

(4) Add the border segment b into B.

(5) Mark $Reg(s)$ as processed.

(6) Find s_{next} whose $Reg(s_{next})$ touches a segment $b \in B$.

(7) If s_{next} exists, then $s = s_{next}$ and go to (3).

(8) Construct $Reg(s_{i+1})$ that is surrounded by segments in B.

(9) Update $Vor_i(S_i)$ and answer $Vor_{i+1}(S_{i+1})$.

Note that, in the Algorithm 3.2, the expected running time for updating $Vor_i(S_i)$ after Algorithm 3.1 in the step (2) is constant in ordinary conditions. Therefore, if we can compute Algorithm 3.1 in constant time, the expected running time to construct a Voronoi diagram by using the incremental method becomes $O(m)$ where m is the number of input points (starting points in this paper).

Algorithms for constructing Voronoi diagrams have been investigated intensively. The problem is proved to be $\Omega(m \log m)$ [2]. And, there is an algorithm whose worst time complexity is $O(m \log m)$, which is the optimal. Though the worst time complexity of the above incremental method is $O(m^2)$, we can improve the expected running time to be $O(m)$ by

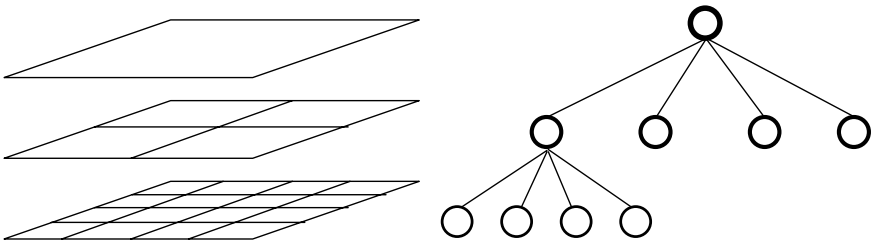

Figure 4: Quaternary Tree Structure

using a quaternary tree bucketing [17, 18]. Therefore, we use Algorithm 3.1 and 3.2 for the data mining system.

Quaternary Bucketing

In the Algorithm 3.1 and 3.2, we used a quaternary tree structure like in Figure 4. First of all, we compute a large rectangle that covers all starting points and query points. Then, we divide the rectangle into four equal-sized subrectangle, recursively. We set the depth of the quaternary tree so that the average number of starting points of a kind in each leaf node is close to one.

Since the width and depth of each leaf node of the tree is same, we can find a leaf node for each point in a constant time. Therefore, bucketing of m starting points and n query points into the leaves can be done in $O(m)$ and $O(n)$ time respectively.

After the bucketing, we assign a representative starting point to each leaf node as a label of the node. A representative starting point of a leaf node is chosen arbitrary from starting points that belongs the node. If there is no starting point in a leaf node, we label the node null. In the efficient method presented in [17, 18], each intermediate node is also labeled with one of the four labels of its children like in Figure 5. Next, all starting points are sorted by breadth-first order of the tree. Then, the method applies Algorithm 3.2 by adding starting points with the order. The expected running time of the method is $O(m)$. We omit the detailed ordering of the starting points and complexity analysis in this paper.

We use the same quaternary tree to find the nearest starting point for each query point.

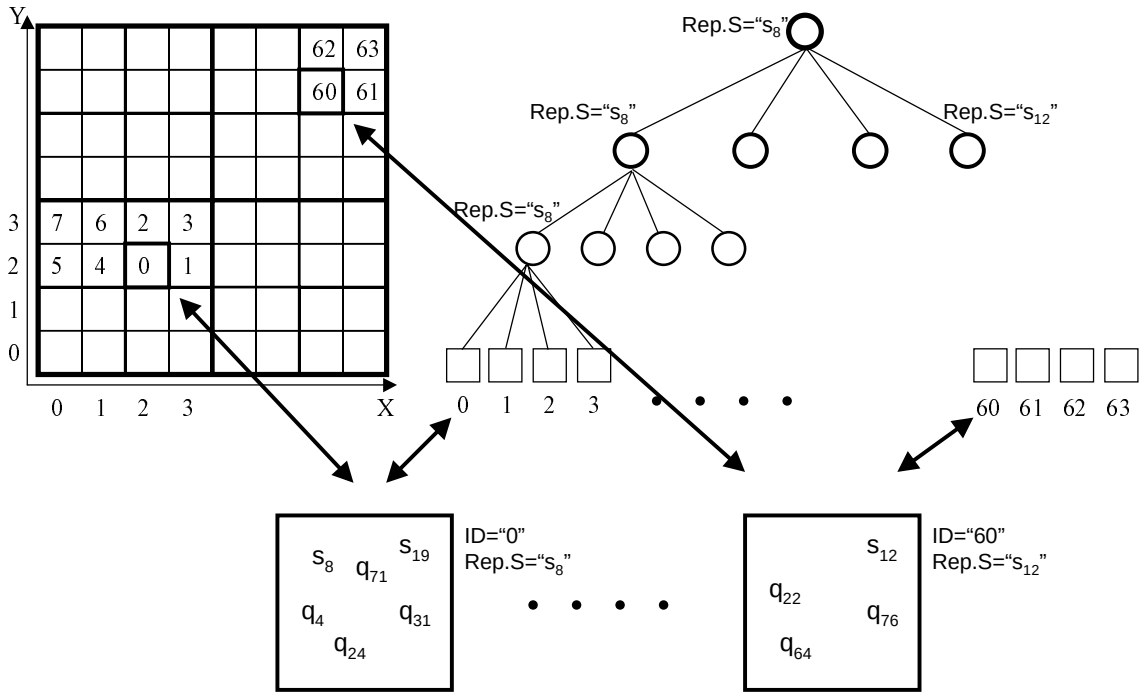

Figure 5: Labeling with Representative Starting Point

For each query point, we search for the nearest starting point from the labeled starting point of the leaf node that the query point lies in. If the label is null, we use the label of its ancestor node. If the nearest point that is found by Algorithm 3.1 is different from the corresponding labeled point, we update the label to the nearest point adaptively. This operation improves the expected running time of Algorithm 3.1 significantly. And, the expected running time for finding the nearest starting point for a query point becomes constant. Therefore, computing an ordered distance / orientation table takes $O(n)$ time by using the quaternary tree and Voronoi diagram.

Note that we need only one quaternary tree to construct all voronoi diagrams and ordered distance / orientation tables.

4 Experiments

4.1 Scalability Examination by Using Synthetic Data

We implemented the proposed mining system and performed several experiments to evaluate the performance. All experiments were done on an IBM IntelliStation which consists of a Pentium II processor running at 450 MHz with 512 KB of L2 cache and 128 MB of real memory.

We generated sets of synthetic starting points and query points with several number of records. All records have an attribute for identification and two coordinate values in the two-dimensional plane. Query points have additional non-spatial features (attributes) which are used for finding optimized distance rules.

We divided the execution time of the mining for optimized distance rules into two parts: *preprocess* and *data mining process*. The preprocess is time taken for constructing Voronoi diagrams by using a quaternary tree. The data mining process is time taken for computing ordered distance tables and optimized distance rules after the preprocess. The *total process* is the sum of the preprocess and the data mining process, i.e., total time taken to compute optimizad distance rules.

We performed following experiments with various number of starting points m and query points n. Each graph of Figure 6, shows relationships between the execution time for a fixed number of query points and various number of starting points.

There are three lines in each graph, which are labeled as *preprocess*, *mining* and *total*. They correspond to execution time of preprocess, data mining process and total process respectively. These notations are also used in other graphs in this section.

There is a gap in the execution time of the preprocesses between $m = 2000$ and $m = 3000$. This gap comes from the change of the depth of the quaternary tree. The mining system adjusts the depth of the quaternary tree so that the average number of starting points in each leaf is close to one. Therefore, the mining system changes the depth according to m. And, when $m = 2048$, the mining system adaptively changes the depth of the tree.

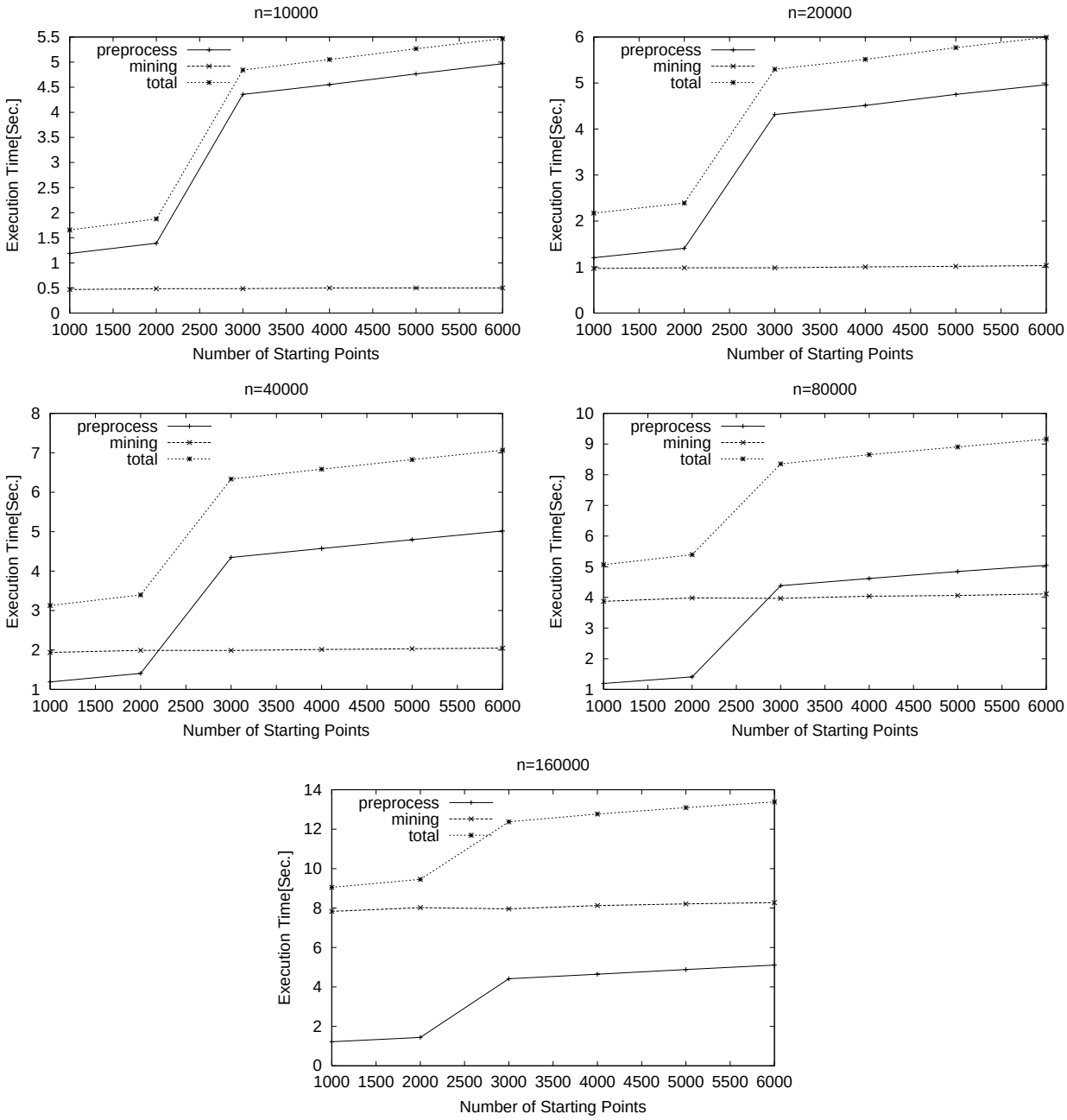

Figure 6: Execution Time Dependence on Number of Starting Point

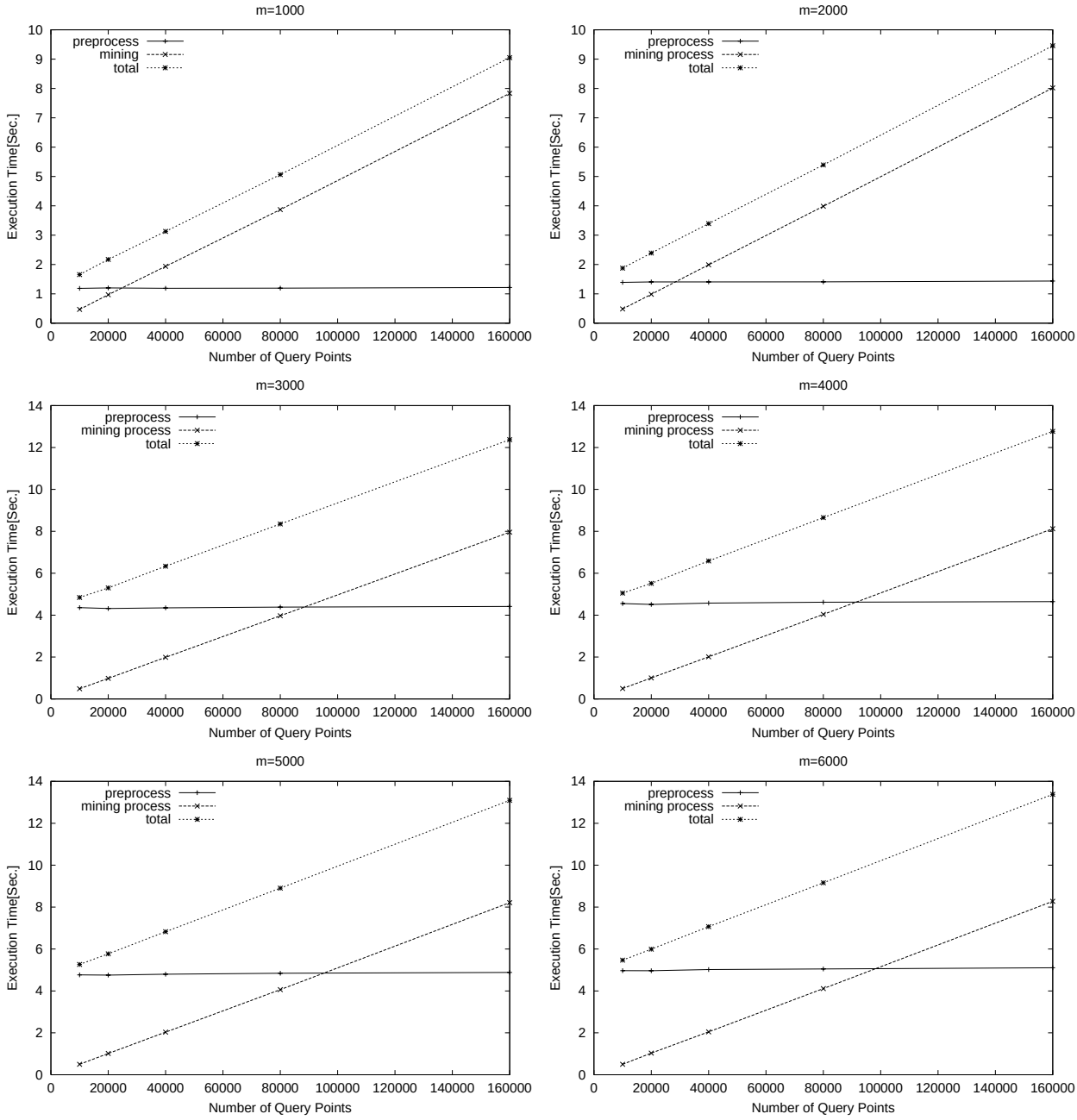

Figure 7: Execution Time Dependence on Number of Query Points

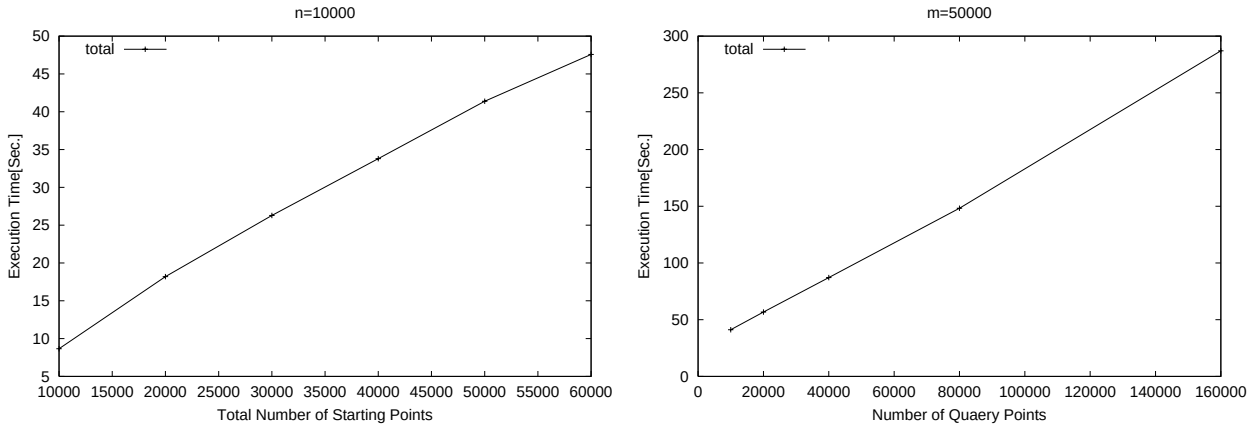

Figure 8: Real Data for Starting Points

Notice that all the execution time of the data mining process in Figure 6 are *constant*. This means that the quaternary tree is effective for data mining process.

Each graph in Figure 7, on the other hand, shows relationships between the execution time for a fixed number of starting points and various number of query points. Because n is fixed in every experiments, the execution time of the preprocesses are constant. The execution time of the data mining process have linear dependence on number of query points.

4.2 Workability Examination for Real Data

In general, real spatial objects in spatial databases are not uniformly distributed in the two-dimensional plane. We used real spatial objects for starting points which are obtained from NTT townpage database in order to examine its workability in practice.

We used several sets of starting points in the real data whose m varies in this experiments. As for query points, we used synthetic data. There are 455 sets of starting points in the real data. Some sets contain only small number of starting points while some sets contain several thousands. Since constant factor is dominant in the performance of the mining system when m is small, we ignored sets whose m is less than 1000 in this experiments to examine the scalability for real data.

The left graph of Figure 8 shows relationships between the total execution time for fixed

number of query points $n = 10000$ and various number of *total* starting points. In this experiment, m is the sum of all sets of starting points. And the total execution time is also the sum of execution time for all non-ignored sets.

The right graph of Figure 8 shows relationships between the total execution time for fixed number of total starting points $m = 50000$, that is, the sum of all sets of starting points, and various number of query points.

These experiments show that our mining system efficiently work for real data.

5 Concluding Remarks

We present a data mining system, which uses efficient algorithms in computational geometry, for finding the optimized distance and / or orientation rules according to specified criteria. For quantitative predicates like distance and orientation, finding such optimized rules is highly demanding. And the experiment results show that our mining system is efficient.

In this paper, we considered distance and orientation between a point and another point. However, in spatial databases, we have to consider other types if spatial objects like lines (connected segments) and polygons. If we have to consider distance and orientation between such spatial objects, problem becomes much complicated. We are now considering such complicated problems for our future works.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 207–216, May 1993.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, 1997.

[3] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24th International Conference on Very Large Data Bases, VLDB*, pages 323–333, 1998.

[4] M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases - focusing techniques for efficient class identification. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases, SSD*, volume 951, pages 67–82. Springer-Verlag, 1995.

[5] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 13–23, June 1996.

[6] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Finding optimal intervals using comptational geometry. In *International Symposium on Algorithm and Computing*, volume 1178 of *Lecture Notes in Computer Science, LNCS*, pages 55–64. Springer-Verlag, 1996.

[7] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining optimized association rules for numeric attributes. *Journal of Computer and System Sciences*, 58:1–12, 1999.

[8] P. J. Green and R. Sibson. Computing dirichlet tessellation in the plane. *The Computer Journal*, 21:168–173, 1978.

[9] R. H. Güting. An introduction to spatial database systems. *VLDB Journal*, 3(4):357–400, Oct. 1994.

[10] J. Han, K. Koperski, and N. Stefanovic. GeoMiner: A system prototype for spatial data mining. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(2):553–556, 1997.

[11] E. M. Knorr and R. T. Ng. Finding aggregate proximity relationships and commonalities in spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 8:884–897, Dec. 1996.

[12] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases, SSD*, volume 951 of *Lecture Notes in Computer Science, LNCS*, pages 47–66. Springer-Verlag, 1995.

[13] Y. Morimoto, T. Fukuda, H. Matsuzawa, T. Tokuyama, and K. Yoda. Algorithms for mining association rules for binary segmentations of huge categorical databases. In *Proceedings of the 24th International Conference on Very Large Data Bases, VLDB*, pages 380–391, 1998.

[14] Y. Morimoto, T. Fukuda, S. Morishita, and T. Tokuyama. Implementation and evaluation of decision trees with range and region splitting. *Constraint*, 2(3/4):401–427, Dec. 1997.

[15] Y. Morimoto, H. Ishii, and S. Morishita. Efficient construction of regression trees with range and region splittng. In *Proceedings of the 23nd International Conference on Very Large Data Bases, VLDB*, pages 166–175, 1997.

[16] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 144–155, 1994.

[17] T. Ohya, M. Iri, and K. Murota. A fast voronoi diagram algorithm with quaternary tree bucketing. *Information Processing Letters*, 18(4):227–231, 1984.

[18] T. Ohya, M. Iri, and K. Murota. Improvements of the incremental method for the voronoi diagram with computational comparison of various algorithms. *Journal of the Operations Research Society of Japan*, 27(4):306–337, 1984.

[19] D. Rhynsburger. Analytic delineation of thiessen polygons. *Geographic Analysis*, 5:138–144, 1973.

[20] W. Wang, J. Yang, and R. R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB*, pages 186–195, 1997.