STING : A Statistical Information Grid Approach to Spatial Data Mining

Wei Wang, Jiong Yang, and Richard Muntz
Department of Computer Science
University of California, Los Angeles
{weiwang, jyang, muntz}@cs.ucla.edu

February 20, 1997

Abstract

Spatial data mining, i.e., discovery of interesting characteristics and patterns that may implicitly exist in spatial databases, is a challenging task due to the huge amounts of spatial data and to the new conceptual nature of the problems which must account for spatial distance. Clustering and region oriented queries are common problems in this domain. Several approaches have been presented in recent years, all of which require at least one scan of all individual objects (points). Consequently, the computational complexity is at least linearly proportional to the number of objects to answer each query. In this paper, we propose a hierarchical statistical information grid based approach for spatial data mining to reduce the cost further. The idea is to capture statistical information associated with spatial cells in such a manner that whole classes of queries and clustering problems can be answered without recourse to the individual objects. In theory, and confirmed by empirical studies, this approach outperforms the best previous method by at least an order of magnitude, especially when the data set is very large.

1 Introduction

In general, spatial data mining, or knowledge discovery in spatial databases, is the extraction of implicit knowledge, spatial relations and discovery of interesting characteristics and patterns that are not explicitly represented in the databases. These techniques can play an important role in understanding spatial data and in capturing intrinsic relationships between spatial and nonspatial data. Moreover, such discovered relationships can be used to present data in a concise manner and to reorganize spatial databases to accommodate data semantics and achieve high performance. Spatial data mining has wide applications in many fields, including GIS Systems, image database exploration, medical imaging, etc.[Che97, Fay96a, Fay96b, Kop96a, Kop96b]

The amount of spatial data obtained from satellite, medical imagery and other sources has been growing tremendously in recent years. A crucial challenge in spatial data mining is the efficiency of spatial data mining algorithms due to the often huge amount of spatial data and the complexity of spatial data types and spatial accessing methods. In this paper, we introduce a new statistical information grid-based method (STING) to efficiently process many common “region oriented” queries on a set of points. Region oriented queries are defined later more precisely but informally, they ask for the selection of regions satisfying certain conditions on density, total area, etc. This paper is organized as follows. We first discuss related work in Section 2. We propose our statistical information grid hierarchical structure and discuss the query types it can support in Sections 3 and 4, respectively. The general algorithm as well as a detailed example of processing a

query are given in Section 5. We analyze the complexity of our algorithm in Section 6. In Section 7, we analyze the quality of STING's result and propose a sufficient condition under which STING is guaranteed to return the correct result. Limiting Behavior of STING is in Section 8 and, in Section 9, we analyze the performance of our method. Finally, we offer our conclusions in Section 10.

2 Related Work

Many studies have been conducted in spatial data mining, such as generalization-based knowledge discovery [Kno96, Lu93], clustering-based methods [Est96, Ng94, Zha96], and so on. Those most relevant to our work are discussed briefly in this section and we emphasize what we believe are limitations which are addressed by our approach.

2.1 Generalization-based Approach

[Lu93] proposed two generalization based algorithms: spatial-data-dominant and non-spatial-data-dominant algorithms. Both of these require that a generalization hierarchy is given explicitly by experts or is somehow generated automatically. (However, such a hierarchy may not exist or the hierarchy given by the experts may not be entirely appropriate in some cases.) The quality of mined characteristics is highly dependent on the structure of the hierarchy. Moreover, the computational complexity is $O(N\log N)$, where N is the number of spatial objects.

Given the above disadvantages, there have been efforts to find algorithms that do not require a generalization hierarchy, that is, to find algorithms that can discover characteristics directly from data. This is the motivation for applying clustering analysis in spatial data mining, which is used to identify regions occupied by points satisfying specified conditions.

2.2 Clustering-based Approach

2.2.1 CLARANS

[Ng94] presents a spatial data mining algorithm based on a clustering algorithm called CLARANS (Clustering Large Applications based upon RANdomized Search) on spatial data. This is the first paper that introduces clustering techniques into spatial data mining problems and it represents a significant improvement on large data sets over traditional clustering methods. However the computational complexity of CLARANS is still high. In [Ng94] it is claimed that CLARANS is linearly proportional to the number of points, but actually the algorithm is inherently at least quadratic. The reason is that CLARANS applies a random search-based method to find an "optimal" clustering. The time taken to calculate the cost differential between the current clustering and one of its neighbors (in which only one cluster *medoid* is different) is linear and the number of neighbors that needs to be examined for the current clustering is controlled by a parameter called *maxneighbor*, which is defined as max(250, 1.25%$K(N - K)$) where K is the number of clusters. This means that the time consumed at each step of searching is $\Theta(KN^2)$. It is very difficult to

estimate how many steps need to be taken to reach the local optimum, but we can certainly say that the computational complexity of CLARANS is $\Omega(KN^2)$. This observation is consistent with the results of our experiments and those mentioned in [Est96] which show that the performance of CLARANS is close to quadratic in the number of points.

Moreover, the quality of the results can not be guaranteed when N is large since randomized search is used in the algorithm. In addition, CLARANS assumes that all objects are stored in main memory. This clearly limits the size of the database to which CLARANS can be applied.

2.2.2 BIRCH

Another clustering algorithm for large data sets, called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), is introduced in [Zha96]. The authors employ the concepts of Clustering Feature and CF tree. Clustering feature is summarizing information about a cluster. CF tree is a balanced tree used to store the clustering features. This algorithm makes full use of the available memory and requires a single scan of the data set. This is done by combining closed clusters together and rebuilding CF tree. This guarantees that the computation complexity of BIRCH is linearly proportional to the number of objects. We believe BIRCH still has one other drawback: This algorithm may not work well when clusters are not "spherical" because it uses the concept of radius or diameter to control the boundary of a cluster[1].

2.2.3 DBSCAN

Recently, [Est96] proposed a density based clustering algorithm (DBSCAN) for large spatial databases. Two parameters Eps and MinPts are used in the algorithm to control the density of normal clusters. DBSCAN is able to separate "noise" from clusters of points where "noise" consists of points in low density regions. DBSCAN makes use of an R* tree to achieve good performance. The authors illustrate that DBSCAN can be used to detect clusters of any shape and can outperform CLARANS by a large margin (up to several orders of magnitude). However, the complexity of DBSCAN is $O(N\log N)$. Moreover, DBSCAN requires a human participant to determine the global parameter Eps. (The parameter MinPts is fixed to 4 in their algorithm to reduce the computational complexity.) Before determining Eps, DBSCAN has to calculate the distance between a point and its kth ($k = 4$) nearest neighbors for all points. Then it sorts all points according to the previous calculated distances and plots the sorted *k-dist* graph. This is a time consuming process. Furthermore, a user has to examine the graph and find the first "valley" of the graph. The corresponding distance is chosen as the value of Eps and the resulting clustering quality is highly dependent on the Eps parameter. When the point set to be clustered is the response set of objects satisfying some qualification, then the determination of Eps must be done each time and the cost of DBSCAN will be higher. (In [Est96], the cost quoted did not include this overhead.)

Moreover, all algorithms described above have the common drawback that they are all query-dependent approaches. That is, the structures used in these approaches are dependent on specific query. They are built once for each query and are generally of no use to answer further queries. Therefore, these approaches need to scan the data sets at least once for each query, which causes

[1] We could not verify this since we do not have BIRCH source code.

the computational complexities of all above approaches to be at least $O(N)$, where N is the number of objects.

In this paper, we propose a statistical information grid-based approach called STING (STatistical INformation Grid) to spatial data mining. The spatial area is divided into rectangular cells. We have several different levels of such rectangular cells corresponding to different resolution and these cells form a hierarchical structure. Each cell at a high level is partitioned to form a number of cells of the next lower level. Statistical information of each cell is calculated and stored beforehand and is used to answer queries. The advantages of this approach are:

- It is a query-independent approach since the statistical information exists independently of queries. It is a summary representation of the data in each grid cell, which can be used to facilitate answering a large class of queries.
- The computational complexity is $O(K)$, where K is the number of grid cells at the lowest level. Usually, $K << N$, where N is the number of objects[2].
- Query processing algorithms using this structure are trivial to parallelize the computing.
- When data is updated, we do not need to recompute all information in the cell hierarchy. Instead, we can do an incremental update.

3 Grid Cell Hierarchy

3.1 Hierarchical Structure

We divide the spatial area into rectangle cells (e.g., using latitude and longitude) and employ a hierarchical structure. Let the root of the hierarchy be at level 1; its children at level 2, etc. A cell in level i corresponds to the union of the areas of its children at level $i + 1$. In this paper each cell (except the leaves) has 4 children and each child corresponds to one quadrant of the parent cell. The root cell at level 1 corresponds to the whole spatial area (which we assume is rectangular for simplicity). The size of the leaf level cells is dependent on the density of objects. As a rule of thumb, we choose a size such that the average number of objects in each cell is in the range from several dozens to several thousands. In addition, a desirable number of layers could be obtained by changing the number of cells that form a higher level cell. In this paper, we will use 4 as the default value unless otherwise specified. In this paper, we assume our space is of two dimensions although it is very easy to generalize this hierarchy structure to higher dimensional models. In two dimensions, the hierarchical structure is illustrated in Figure 1.

[2] Some strategies can be applied when constructing the hierarchical structure to ensure $K \leq N$, which are beyond the scope of this paper.

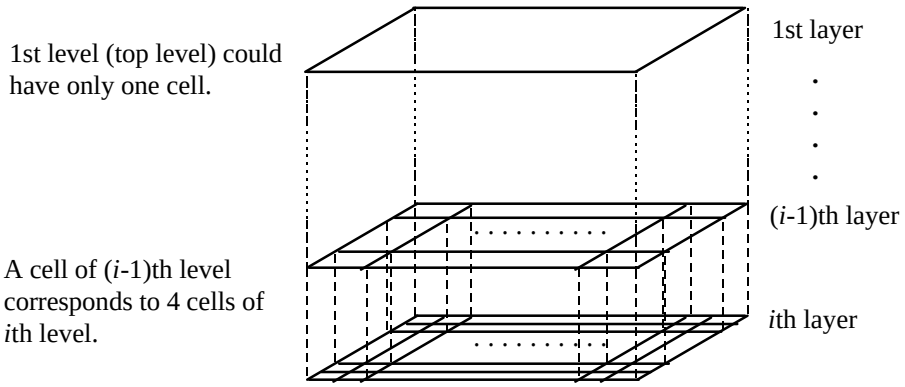

Figure 1. Hierarchical Structure

For each cell, we have attribute-dependent and attribute-independent parameters. The attribute-independent parameter is:

- *n* — number of objects (points) in this cell

As for the attribute-dependent parameters, we assume that for each object, its attributes have numerical values. (We will address the categorical case in future research.) For ***each*** numerical attribute, we have the following five parameters for each cell:

- *m* — mean of all values in this cell
- *s* — standard deviation of all values of the attribute in this cell
- *min* — the minimum value of the attribute in this cell
- *max* — the maximum value of the attribute in this cell
- *distribution* — the type of distribution that the attribute value in this cell follows

The parameter *distribution* is of enumeration type. Potential distribution types are: normal, uniform, exponential, and so on. The value NONE is assigned if the distribution type is unknown. The distribution type will determine a "kernel" calculation in the generic algorithm as will be discussed in detail shortly.

3.2 Parameter Generation

We generate the hierarchy of cells with their associated parameters when the data is loaded into the database. Parameters *n*, *m*, *s*, *min*, and *max* of bottom level cells are calculated directly from data. The value of *distribution* could be either assigned by the user if the distribution type is known before hand or obtained by hypothesis tests such as χ^2-test. Parameters of higher level cells can be easily calculated from parameters of lower level cell. Let *n*, *m*, *s*, *min*, *max*, *dist* be parameters of current cell and n_i, m_i, s_i, min_i, max_i, and $dist_i$ be parameters of corresponding lower level cells, respectively. The *n*, *m*, *s*, *min*, and *max* can be calculated as follows.

$$n = \sum_i n_i$$

$$m = \frac{\sum_i m_i n_i}{n}$$

$$s = \sqrt{\frac{\sum_i (s_i^2 + m_i^2) n_i}{n} - m^2}$$
$$min = \min_i (min_i)$$
$$max = \max_i (max_i)$$

The determination of *dist* for a parent cell is a bit more complicated. First, we set *dist* as the distribution type followed by most points in this cell. This can be done by examining $dist_i$ and n_i. Then, we estimate the number of points, say *confl*, that conflict with the distribution determined by *dist*, *m*, and *s* according to the following rule:

1. If $dist_i \neq dist$, $m_i \approx m$ and $s_i \approx s$, then *confl* is increased by an amount of n_i;
2. If $dist_i \neq dist$, but either $m_i \approx m$ or $s_i \approx s$ is not satisfied, then set *confl* to *n* (This enforces *dist* will be set to NONE later);
3. If $dist_i = dist$, $m_i \approx m$ and $s_i \approx s$, then *confl* is increased by 0;
4. If $dist_i = dist$, but either $m_i \approx m$ or $s_i \approx s$ is not satisfied, then *confl* is set to *n*.

Finally, if $\frac{confl}{n}$ is greater than a threshold *t* (This threshold is a small constant, say 0.05, which is set before the hierarchical structure is built), then we set *dist* as NONE; otherwise, we keep the original type. For example, the parameters of lower level cells are as follows.

i	1	2	3	4
n_i	100	50	60	10
m_i	20.1	19.7	21.0	20.5
s_i	2.3	2.2	2.4	2.1
min_i	4.5	5.5	3.8	7
max_i	36	34	37	40
$dist_i$	NORMAL	NORMAL	NORMAL	NONE

Table 1: Parameters of Children Cells

Then the parameters of current cell will be

$n = 220$
$m = 20.27$
$s = 2.37$
$min = 3.8$
$max = 40$
$dist$ = NORMAL

The distribution type is still NORMAL based on the following: Since there are 210 points whose distribution type is NORMAL, *dist* is first set to NORMAL. After examining $dist_i$, m_i, and s_i of each lower level cell, we find out $confl = 10$. So, *dist* is kept as NORMAL ($\frac{confl}{n} = 0.045 < 0.05$).

We only need to go through the data set once in order to calculate the parameters associated with the grid cells at the bottom level, the overall compilation time is linearly proportional to the number of objects with a small constant factor. (And only has to be done once — not for each query.) With

this structure in place, the response time for a query is much faster since it is $O(K)$ instead of $O(N)$. We will analyze performance in more detail in later sections.

4 Query Types

If the statistical information stored in the STING hierarchical structure is not sufficient to answer a query, then we have recourse to the underlying database. Therefore, we can support any query that can be expressed by the SQL-like language described later in this section. However, the statistical information in the STING structure can answer many commonly asked queries very efficiently and we often do not need to access the full database. Even when the statistical information is not enough to answer a query, we can still narrow the set of possible choices.

STING can be used to facilitate several kinds of spatial queries. The most commonly asked query is region query which is to select regions that satisfy certain conditions (Ex1). Another type of query selects regions and returns some function of the region, e.g., the range of some attributes within the region (Ex2). We extend SQL so that it can be used to describe such queries. The formal definition is in Appendix. The following are several query examples.

Ex1. Select the maximal regions that have at least 100 houses per unit area and at least 70% of the house prices are above $400K and with total area at least 100 units with 90% confidence.

```
SELECT REGION
FROM house-map
WHERE DENSITY IN (100, ∞)
AND price RANGE (400000, ∞) WITH PERCENT (0.7, 1)
AND AREA (100, ∞)
AND WITH CONFIDENCE 0.9
```

Ex2. Select the range of age of houses in those maximal regions where there are at least 100 houses per unit area and at least 70% of the houses have price between $150K and $300K with area at least 100 units in California.

```
SELECT RANGE(age)
FROM house-map
WHERE DENSITY IN (100, ∞)
AND price RANGE (150000, 300000) WITH PERCENT (0.7, 1)
AND AREA (100, ∞)
AND LOCATION California
```

5 Algorithm

With the hierarchical structure of grid cells on hand, we can use a top-down approach to answer spatial data mining queries. For each query, we begin by examining cells on a high level layer. Note that it is not necessary to start with the root; we may begin from an intermediate layer (but we do not pursue this minor variation further due to lack of space).

Starting with the root, we calculate the likelihood that this cell is relevant to the query at some confidence level using the parameters of this cell (exactly how this is computed is described later). This likelihood can be defined as the proportion of objects in this cell that satisfy the query conditions. (If the distribution type is NONE, we estimate the likelihood using some distribution-free techniques instead.) After we obtain the confidence interval, we label this cell to be *relevant* or *not relevant* at the specified confidence level. When we finish examining the current layer, we proceed to the next lower level of cells and repeat the same process. The only difference is that instead of going through all cells, we only look at those cells that are children of the *relevant* cells of the previous layer. This procedure continues until we finish examining the lowest level layer (bottom layer). In most cases, these *relevant* cells and their associated statistical information are enough to give a satisfactory result to the query. Then, we find all the regions formed by *relevant* cells and return them. However, in rare cases (People may want very accurate result for special purposes, e.g. military), this information are not enough to answer the query. Then, we need to retrieve those data that fall into the *relevant* cells from database and do some further processing.

After we have labeled all cells as *relevant* or *not relevant*, we can easily find all regions that satisfy the density specified by a breadth-first search. For each *relevant* cell, we examine cells within a certain distance (how to choose this distance is discussed below) from the center of current cell to see if the average density within this small area is greater than the density specified. If so, this area is marked and all *relevant* cells we just examined are put into a queue. Each time we take one cell from the queue and repeat the same procedure except that only those *relevant* cells that are not examined before are enqueued. When the queue is empty, we have identified one region. The distance we use above is calculated from the specified density and the granularity of the bottom level cell. The distance $d = \max(l, \sqrt{\frac{f}{\pi c}})$ where l, c, and f are the side length of bottom layer cell, the specified density, and a small constant number set by STING (It does not vary from a query to another), respectively. Usually, l is the dominant term in $\max(l, \sqrt{\frac{f}{\pi c}})$. As a result, this distance can only reach the neighbor cells. In this case, we just need to examine neighboring cells and find regions that are formed by connected cells. Only when the granularity is very small, this distance could cover a number of cells. In this case, we need to examine every cell within this distance instead of only neighboring cells.

For example, if the objects in our database are houses and price is one of the attributes, then one kind of query could be "Find those regions with area at least A where the number of houses per unit area is at least c and at least β% of the houses have price between a and b with $(1 - \alpha)$ confidence" where $a < b$. Here, a could be $-\infty$ and b could be $+\infty$. This query can be written as

SELECT REGION
FROM house-map
WHERE DENSITY IN [c, ∞)
AND price RANGE [a, b] WITH PERCENT [β%, 1]
AND AREA [A, ∞)
AND WITH CONFIDENCE 1 - α

We begin from the top layer that has only one cell and stop at the bottom level. Assume that the price in each bottom layer cell is approximately normally distributed. (For other distribution types the idea is essentially the same except that we use different distribution function and lookup table.) Note that price in a higher level cell could have distribution type as NONE.

For each cell, if the distribution type is normal, we first calculate the proportion of houses whose price is within the range [*a*, *b*]. The probability that a price is between *a* and *b* is

$$\begin{aligned}\hat{p} &= P(a \le price \le b) \\ &= P(\frac{a-m}{s} \le \frac{price-m}{s} \le \frac{b-m}{s}) \\ &= P(\frac{a-m}{s} \le Z \le \frac{b-m}{s}) \\ &= \Phi(\frac{b-m}{s}) - \Phi(\frac{a-m}{s})\end{aligned}$$

where *m* and *s* are the mean and standard deviation of all prices in this cell respectively. Since we assume all prices are independent given the mean and variance, the number of houses with price between *a* and *b* has a binomial distribution with parameters *n* and $\hat{p}$, where *n* is the number of houses. Now we consider the following cases according to *n*, $n\hat{p}$, and $n(1 - \hat{p})$.

1. When $n \le 30$, we can use binomial distribution directly to calculate the confidence interval of the number of houses whose price falls into [*a*, *b*], and divide it by *n* to get the confidence interval for the proportion.
2. When $n > 30$, $n\hat{p} \ge 5$, and $n(1 - \hat{p}) \ge 5$, the proportion that the price falls in [*a*, *b*] has a normal distribution $N(\hat{p}, \sqrt{\hat{p}(1-\hat{p})/n})$ approximately. Then $100(1 - \alpha)\%$ confidence interval of the proportion is $\hat{p} \pm z_{\alpha/2}\sqrt{\hat{p}(1-\hat{p})/n} = [p_1, p_2]$.
3. When $n > 30$ but $n\hat{p} < 5$, the Poisson distribution with parameter $\lambda = n\hat{p}$ is approximately equal to the binomial distribution with parameters *n* and $\hat{p}$. Therefore, we can use the Poisson distribution instead.
4. When $n > 30$ but $n(1 - \hat{p}) < 5$, we can calculate the proportion of houses (*X*) whose price is not in [*a*, *b*] using Poisson distribution with parameter $\lambda = n(1 - \hat{p})$, and $1 - X$ is the proportion of houses whose price is in [*a*, *b*].

For a cell, if the distribution type is NONE, we can estimate the proportion range $[p_1, p_2]$ that the price falls in [*a*, *b*] by some distribution-free techniques, such as Chebyshev's inequality [Dev91].

1. If $m \notin [a, b]$, then $[p_1, p_2] = \left[0, \min\left(\max\left(\frac{s^2}{(a-m)^2}, \frac{s^2}{(b-m)^2}\right), 1\right)\right]$;
2. If $m = a$ or $m = b$, then $[p_1, p_2] = [0, 1]$;
3. If $m \in (a, b)$, then $[p_1, p_2] = \left[\max\left(1 - \frac{s^2}{(a-m)^2}, 1 - \frac{s^2}{(b-m)^2}, 0\right), 1\right]$.

Once we have the confidence interval or the estimated range $[p_1, p_2]$, we can label this cell as *relevant* or *not relevant*. Let S be the area of cells at bottom layer. If $p_2 \times n < S \times c \times \beta\%$, we label this cell as *not relevant*; otherwise, we label it as *relevant*.

Each time when we finish examining a layer, we go down one level and only examine those cells that form the *relevant* cells at higher layer. After we labeled the cells at bottom layer, we scan those *relevant* cells and return those regions formed by at least $\lceil A/S \rceil$ adjacent *relevant* cells. This can be done in $O(K)$ time.

The above algorithm is summarized in Figure 2.

Statistical Information Grid-based Algorithm:

1. Determine a layer to begin with.
2. For each cell of this layer, we calculate the confidence interval (or estimated range) of probability that this cell is relevant to the query.
3. From the interval calculated above, we label the cell as *relevant* or *not relevant.*
4. If this layer is the bottom layer, go to Step 6; otherwise, go to Step 5.
5. We go down the hierarchy structure by one level. Go to Step 2 for those cells that form the *relevant* cells of the higher level layer.
6. If the specification of the query is met, go to Step 8; otherwise, go to Step 7.
7. Retrieve those data fall into the *relevant* cells and do further processing. Return the result that meet the requirement of the query. Go to Step 9.
8. Find the regions of *relevant* cells. Return those regions that meet the requirement of the query. Go to Step 9.
9. Stop.

Figure 2. STING Algorithm

6 Analysis of the STING Algorithm

In above algorithm, Step 1 takes constant time. Steps 2 and 3 require a constant time for each cell to calculate the confidence interval or estimate proportion range and also a constant time to label the cell as *relevant* or *not relevant*. This means that we need constant time to process each cell in Steps 2 and 3. The total time is less than or equal to the total number of cells in our hierarchical structure. Notice that the total number of cells is $1.33K$, where K is the number of cells at bottom layer. We obtain the factor 1.33 because the number of cells of a layer is always one-forth of the number of cells of the layer one level lower. So the overall computation complexity on the grid hierarchy structure is $O(K)$. Usually, the number of cells needed to be examined is much less, especially when many cells at high layers are *not relevant*. In Step 8, the time it takes to form the regions is linearly proportional to the number of cells. The reason is that for a given cell, the number of cells need to be examined is constant because both the specified density and the granularity can be regarded as constants during the execution of a query and in turn the distance is also a constant since it is determined by the specified density. Since we assume each cell at bottom layer usually has several dozens to several thousands objects, $K << N$. So, the total complexity is still $O(K)$.Usually, we do not need to do Step 7 and the overall computational complexity is $O(K)$.

In the extreme case that we need to go to Step 7, we still do not need to retrieve all data from database. Therefore, the time required in this step is still less than linear. So, this algorithm outperforms other approaches greatly.

7 Quality of STING

STING makes use of statistical information to approximate the expected results of query. Therefore, it could be imprecise since data points can be arbitrarily located. However, under one of the following two conditions, STING can guarantee the accuracy of its result. Let A and c be the minimum area and density specified by query, respectively. Let R and l be a region satisfying the conditions specified by the query and the side length of bottom level cell, respectively.

Definition 1. Let F be a region. The *width* of F is defined as the side length of the maximum square that can fit in F.

1. Let W be the width of R. If $W^2 - 4(\lceil W/l \rceil + 1)l^2 \geq A$, then R must be returned by STING. The reason is that the square with side length W covers more than $W^2/l^2 - 4(\lceil W/l \rceil + 1)$ bottom level cells entirely. Since all these cells will be detected, STING is able to return R.

Definition 2. Let S_1 and S_2 be two squares. The *distance* between S_1 and S_2 is defined as the maximum distance between vertices of S_1 and S_2.

2. If at least $\lceil A/l^2 \rceil$ squares with side length of $2\sqrt{2}l$ can fit in R and there exists a tree on those squares such that the distance between the parent square and its child is within $\sqrt{\frac{f}{\pi c}}$ where f is the small constant set by the system, then R must be returned by STING. The reason is that each of those squares covers at least one bottom level cell entirely. Therefore, STING is able to discover R.

The above is the sufficient condition for STING to return accurate results. However, in most of other cases, STING is also able to return correct answers with high confidence. The worst case scenario for STING would be a cluster of points right at the corners of four cells in the center of the map. We use the following strategy to solve this problem.

1. We make the size of bottom level cell near zero such that each bottom level cell contains at most one data point if no two points collocate. We only instantiate a cell if there is at least one data point in it.
2. We intelligently construct the hierarchical structure such that the number of instantiated cells in a higher layer is at most half of that in one level lower.
3. We only keep a certain number of top levels on line and the rest layers are kept off-line. If an off-line layer is needed, we can dynamically load it in. However, users rarely requires such precision.

Pursuit of this extension is beyond the scope of this paper and will be dealt with in future work.

8 Limiting Behavior of STING is Equivalent to DBSCAN

The regions returned by STING are an approximation of the result by DBSCAN. As the granularity approaches zero, the regions returned by STING approach the result of DBSCAN. In order to compare to DBSCAN, we only use the number of points here since DBSCAN can only cluster points according to their spatial location. (i.e., we do not consider conditions on other attributes.) DBSCAN has two parameters: Eps and MinPts. (Usually, MinPts is fixed to *k*.) In our case, STING has only one parameter: the density *c*. We set $c = \frac{\text{MinPts}+1}{\pi\text{Eps}^2} = \frac{k+1}{\pi\text{Eps}^2}$ in order to approximate the result of DBSCAN. The reason is that the density of any area inside the clusters detected by DBSCAN is at least $\frac{\text{MinPts}+1}{\pi\text{Eps}^2}$ since for each core point there are at least MinPts points (excluding itself) within distance Eps. In STING, for each cell, if $n < S \times c$, then we label it as *not relevant*; otherwise, we label it as *relevant* where *n* and *S* are the number of points in this cell and the area of bottom layer cell, respectively. When we form the regions from *relevant* cells, the examining distance is set to be $d = \max(l, \sqrt{\frac{k+1}{\pi c}})$. When the granularity is very small, $\sqrt{\frac{k+1}{\pi c}}$ becomes the dominant term. As the granularity approaches zero, the area of each cell at bottom layer goes to zero. So, if there is at least one point in a cell, this cell will be labeled as *relevant*. Now what we need to do is to form the region to be returned according to distance *d* and density *c*. We can see that $d = \sqrt{\frac{k+1}{\pi c}} = \sqrt{\frac{k+1}{\pi \frac{k+1}{\pi\text{Eps}^2}}} = \text{Eps}$. For each *relevant* cell, we examine the area around it (within distance *d*) to see if the density is greater than *c*. This is equivalent to check if the number of points (including itself) within this area is greater than $c \times \pi d^2 = k + 1$. As a result, the result of STING approaches that of DBSCAN when the granularity approaches zero.

9 Performance

We run several tests to evaluate the performance of STING. The following tests are run on a SPARC 10 machine with Solaris 2.4 operating system (192 MB memory).

9.1 Performance Comparison of Two Distributions

To obtain performance metric of STING, we implemented the house-price example discussed in Section 5. Ex1 is the query that we posed. We generated two data sets, both of which have 100,000 data points (houses). The hierarchical structure has seven layers in this test. First, we generate a data set (DS1) such that the price is normally distributed in each cell (with similar mean). The hierarchical structure generation time is 9.8 seconds. (Generation needs to be done once for each data set. All the queries for the same data set can use the same structure. Therefore, we do not need to generate it for each query.) It takes STING 0.20 second to answer the query given the STING

structure exists. The expected result and the result returned by STING are in Figure 3a and 3b, respectively.

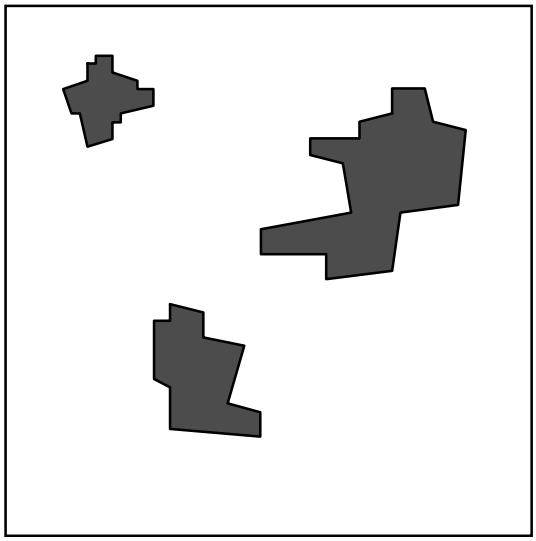

Figure 3a. Expected result of DS1

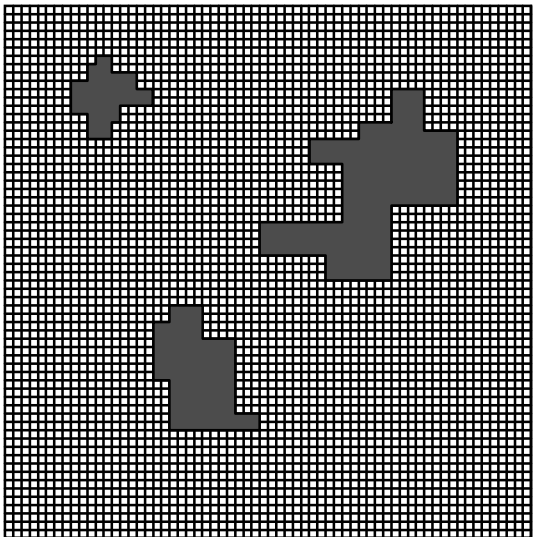

Figure 3b. STING's result of DS1

From Figure 3a and 3b, we can see that STING's result is very close to the expected one. In the second data set (DS2), the prices in each bottom layer cell follow a normal distribution (with different mean) but they do not follow any known distribution at higher levels. The hierarchical structure generation time is 9.7 seconds. It takes STING 0.22 second to answer the query. The expected result and the result returned by STING are in Figure 4a and 4b, respectively.

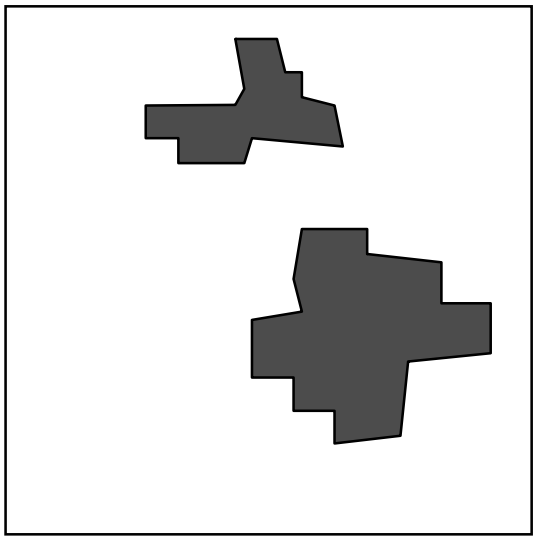

Figure 4a. Expected result of DS2

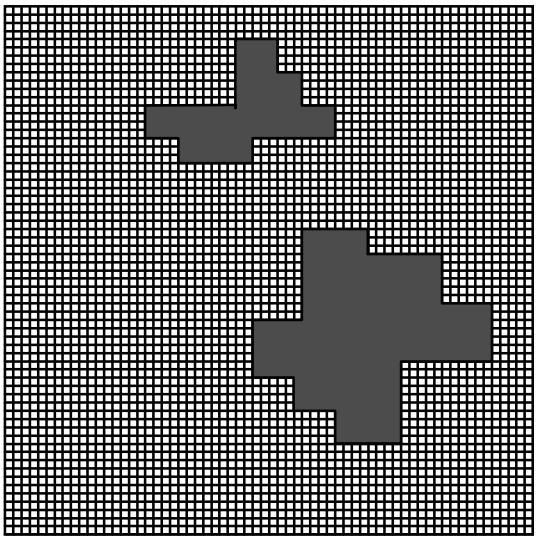

Figure 4b. STING's result of DS2

Once again, we can see that the STING's result is very closed to the expected one.

9.2 Benchmark Result

Currently, clustering based approaches are an important category of spatial data mining problems. Three extant systems are CLARANS [Ng94], BIRCH [Zha96], and DBSCAN [Est96]. We compare the performance of these three with STING.

In the following tests, we only compare the time for clustering. However, if the clustering data is the result of some query, then all other algorithms (other than STING) have at least three phases:

1. Find query response.
2. Build auxiliary structure.
3. Do clustering.

The reported numbers for the other methods do not include computation of Phase 1, but STING only takes one step to answer the whole query. Therefore, STING actually compares better than that the measurements presented here indicate.

We use the benchmark chosen by Ester M. et al. in [Est96], namely SEQUOIA 2000 [Sto93], to compare the performance of STING and other approaches. We successfully ran CLARANS and STING with data size between 1252 and 12512. STING has generation time and query time. The generation time is the time consumed to generate the hierarchical structure and the query time is the time used to answer a specific query. In the test, the STING hierarchy structure has six layers.

Due to unavailability of DBSCAN source code, we are unable to run this algorithm. We discovered that CLARANS is approximately 15 times faster in our configuration than in the configuration specified in [Est96] for all data sizes. We estimate that DBSCAN also runs roughly 15 times faster and show the estimated running time of DBSCAN in the following table as a function of point set cardinality. All times are in units of seconds.

Number of Points	1256	2503	3910	5213	6256	12512
CLARANS	49	200	457	785	1238	5538
DBSCAN (projected)	0.2	0.4	0.7	1.0	1.2	2.86
STING (query)	0.1	0.11	0.11	0.12	0.12	0.14
STING (generation)	1.25	1.32	1.40	1.48	1.55	1.62

Table 2: Performance tests for CLARANS, DBSCAN, and STING

Furthermore, BIRCH outperforms CLARANS about 20 to 30 times [Zha96]. So STING will also outperform BIRCH by a very large margin. We plot the query response time for DBSCAN and STING in Figure 5 because DBSCAN is the fastest one among all existing algorithms.

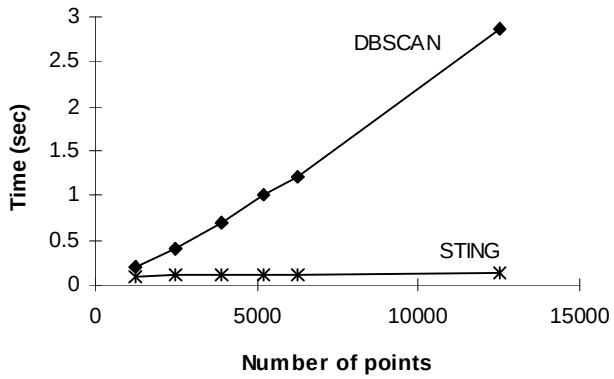

Figure 5. Performance Comparison between STING and DBSCAN

10 Conclusion

In this paper, we present a statistical information grid-based approach to spatial data mining. It has much less computational cost than other approaches. The I/O cost is low since we can usually keep the STING data structure in memory. Both of these will speed up the processing of spatial data query tremendously. In addition, it offers us an opportunity for parallelism (STING is trivially parallelizable). All these advantages benefit from the hierarchical structure of grid cells and the statistical information associated with them.

References

[Che97] M. S. Chen, J. Han, P. S. Yu. Data mining: an overview from database perspective. to appear in *IEEE Transactions on Knowledge and data Engineering*, 1997.

[Dev91] J. L. Devore. *Probability and Statistics for Engineering and the Sciences*, 3rd edition. Brooks/Cole Publishing Company, Pacific Grove, California, 1991.

[Est95] M. Ester, H. P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. *Proc. 4th Int. Symp. on Large Spatial Databases* (*SSD'95*), pp. 67-82, Poland, Maine, August 1995.

[Est96] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining* (*KDD-96*), pp. 226-231, Portland, OR, USA, August 1996.

[Fay96a] U. Fayyad, G. P.-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, Vol. 17 No. 3, pp. 37-54, Fall 1996.

[Fay96b] U. Fayyad, G. P.-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, Menlo Park, CA, 1996.

[Fot94] S. Fotheringham and P. Rogerson. *Spatial Analysis and GIS*. Taylor and Francies, 1994

[Kno96] E. M. Knorr and R. Ng. Extraction of spatial proximity patterns by concept generalization. *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining* (*KDD-96*), pp. 347-350, Portland, OR, USA, August 1996.

[Kop96a] K. Koperski, J. Adhikary, and J. Han. Spatial data mining: progress and challenges. *SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery* (*DMKD'96*), Montreal, Canada, June 1996.

[Kop96b] K. Koperski and J. Han. Data mining methods for the analysis of large geographic databases. *Proc. 10th Annual Conf. on GIS*. Vancouver, Canada, March 1996.

[Lu93] W. Lu, J. Han, and B. C. Ooi. Discovery of general knowledge in large spatial databases. *Proc. Far East Workshop on Geographic Information Systems*, pp. 275-289, Singapore, June 1993.

[Ng94] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. *Proc. 1994 Int. Conf. Very Large Databases*, pp. 144-155, Santiago, Chile, September 1994.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[Sto93] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The SEQUOIA 2000 storage benchmark. *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pp. 2-11, Washington DC, 1993.

[Zha96] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pp. 103-114, Montreal, Canada, June 1996.

Appendix

The following is the specification of our extended SQL in BNF notation.

<query> ::= <region-query> | <object-query> | <func-query>
<region-query> ::= SELECT REGION
FROM <from-clause>
WHERE <region-conds>
<object-query> ::= SELECT object
FROM <from-clause>
WHERE <object-conds>
<attr-query> ::= SELECT <attr-funcs>
FROM <from-clause>
WHERE <attr-conds>
<from-clause> ::= <relations> | <classes>
<relations> ::= relation-name | relation-name, <relations>
<classes> ::= class-name | class-name, <classes>
<region-conds> ::= <region-cond> | <region-cond> AND <region-conds>
<region-cond> ::= <density> | <func> | <area> | <location> | <confidence>
<object-conds> ::= <object-cond> | <object-cond> AND <object-conds>
<object-cond> ::= <obj-func> | <location>
<attr-funcs> ::= <attr-func> | <attr-func>, <attr-funcs>
<attr-func> ::= attr-name | <stat-func>(attr-name)
<stat-func> ::= MAX | MIN | RANGE | AVERAGE | SUM | COU NT | ...
<func-conds> ::= <region-conds> | <object-conds>
<density> ::= DENSITY IN <left-paren>number, number<right-paren>
<func> ::= <obj-func> [WITH PERCENT
<left-paren>percentage, percentage<right-paren>]
<obj-func> ::= <attr-func> RANGE <left-paren>number, number<right-paren>
<area> ::= AREA <left-paren>number, number<right-paren>
<location> ::= LOCATION <namelist> | LOCATION <polygonlist>
<confidence> ::= WITH CONFIDENCE percentage
<namelist> ::= name | name; <namelist>
<polygonlist> ::= <polygon> | <polygon>; <polygonlist>
<polygon> ::= <points>
<points> ::= <point> | <point>, <points>
<point> ::= (coordinate, coordinate)
<left-paren> ::= “[” | “(”
<right-paren> ::= “]” | “)”