An Approach to Active Spatial Data Mining Based on Statistical Information

Wei Wang, Jiong Yang
T.J. Watson Research Centers
IBM Research Division
{ww1, jiyang}@us.ibm.com

Richard Muntz
Department of Computer Science
University of California, Los Angeles
muntz@cs.ucla.edu

Revised April 28, 2000

Abstract

Spatial data mining presents new challenges due to the large size of spatial data, the complexity of spatial data types, and the special nature of spatial access methods. Most research in this area has focused on efficient query processing of static data. This paper introduces an active spatial data mining approach that extends the current spatial data mining algorithms to efficiently support user-defined triggers on dynamically evolving spatial data. To exploit the locality of the effect of an update and the nature of spatial data, we employ a hierarchical structure with associated statistical information at the various levels of the hierarchy and decompose the user-defined trigger into a set of sub-triggers associated with cells in the hierarchy. Updates are suspended in the hierarchy until their cumulative effect might cause the trigger to fire. It is shown that this approach achieves three orders of magnitude improvement over the naive approach that re-evaluate the condition over the database for each update, while both approaches produce the same result without any delay. Moreover, this scheme can support incremental query processing as well.

1 Introduction

Spatial data mining, i.e., discovery of interesting characteristics and patterns that may implicitly exist in spatial databases, plays an important role in understanding spatial data and in capturing intrinsic relationships between spatial and non-spatial data. Efficiency is a crucial challenge in spatial data mining due to the large size of spatial data and the complexity of spatial data types and spatial access methods.

There have been many contributions in this field during recent years [Kno96a] [Kno96b] [Kop96] [Est97] [Gal97] [Han97] [Hor97] [Wan97] [Est98a] [Est98b] [Gru98] [Zai98]. However, most approaches have focused on issues in processing spatial data mining queries. Our focus in this paper is to extend current spatial data mining techniques to support user-defined triggers, i.e., *active* spatial data mining. In this paper, we assume point objects unless otherwise specified. Each object has a spatial attribute, its location, and some non-spatial attributes. In many applications, clusters formed by objects with some specified attribute values are the main features of interest. Some examples are the following.

1. *Military Deployment*. For example, armor deployment in some region can be located via satellite images. The movement of armored vehicles can be traced and if specified patterns of concentration or movement are detected, further investigation may be triggered.

2. *Cellular Phone Service*. Each mobile object has an ID, associated personal information, and can move from one area to another. Patterns in time, length, distance, and location of phone calls can be mined. Such knowledge can be used to dynamically allocate bandwidth for better service and price policy planning for maximum profit.

3. *Situation Awareness and Emergency Response*. Situation awareness and emergency response are two important information centered applications that require support for a large number of geographically distributed mobile users collaborating on a common mission and with interest in a common situation domain. The spatial location of a user plays a dominant rule in determining the user profile. Clustering of users performing the same task in close spatial proximity can be used to balance the tradeoff between minimizing the overall bandwidth requirement and maximizing the relevance of multicast channels to users. Triggers specified in terms of changes to clusters (area, density, location, etc.) may be used to signal the need to adjust routing assignments.

Introducing spatial data mining triggers has the following advantages.

1. From the user's point of view, it is not necessary to submit the same query repeatedly in order to monitor the appearance of some condition. Instead, he (or she) may specify a trigger and code the pattern as the trigger condition. Moreover, instead of waiting until the next execution of the same query, interesting patterns can be detected immediately as they appear if triggers are used. In addition, if the pattern only exists for a short period between two executions of the query, the user may miss the pattern using the reposting query method.

2. From the system's point of view, it is usually much more efficient to handle a trigger incrementally than re-execute the same query on the entire database many times.

3. Data mining tasks involving special characteristics associated with spatial data, such as cluster emergence, movement, splitting, merging, and vanishing, cannot be supported by traditional database triggers efficiently (if

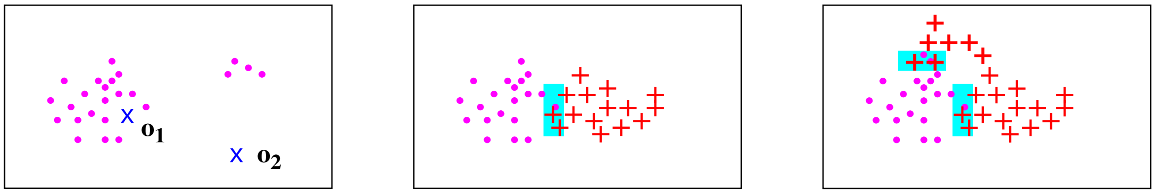
at all). This is due to the fact that the class membership of an object is not only determined by its non-spatial attributes but also by the attributes of objects in its neighborhood. As an extension of both spatial data mining techniques and active rules, spatial data mining triggers are designed to handle such complicated tasks efficiently.

In this paper, we introduce an approach to active spatial data mining, called STING+, which takes advantage of the rich research results of active database systems and the efficient algorithms in STING [Wan97] for *passive* spatial data mining. Instead of the traditional *Event-Condition-Action* paradigm [Wid96] [Zan97], triggers in STING+ do not require the *Event* specification and thus fall into the *Condition-Action* paradigm [Wid96]. Any condition allowed in a spatial data mining query can be specified as the *Condition* of a spatial data mining trigger. This enables the user to specify a complicated condition and the action taken upon satisfaction of some condition without considering which event(s) might cause this condition to become true. This feature is important because a data mining task is usually very complicated. It is very difficult (if not impossible) for a user to specify all events that might affect the trigger condition. In contrast, in STING+, all events that may cause the trigger condition to be satisfied are accounted for by the system during trigger evaluation.

Evaluating a user-defined trigger T usually involves two aspects, where C_T is the trigger condition specified in T. (1) Find a set of composite events[1]. $E(s)$ such that if C_T transitions from false to true then this transition must have occurred due to some composite event in $E(s)$, where s and $E(s)$ are the state of the database, and a subset of all composite events that can cause C_T to become true, respectively. (2) Each time some composite event in $E(s)$ occurs, check the status (false or true) of C_T given that C_T was false previously. In general, $E(s)$ is a function of the database state s. In spatial databases, object insertion, deletion, and update are primitive events to form a composite event. For example, a composite event could be 100 object insertions in a specified region. The effect of such a composite event depends largely on conditions in such a region. For example, as illustrated in Figure 1(a), deleting object o_1 from the region which a cluster occupies could potentially cause this cluster to shrink whereas deleting object o_2 from some other place has no effect on this cluster. As a side effect of the occurrence of some composite event, the set of composite events $E(s)$ that could cause C_T to transition from false to true might also evolve over time. For example, the dot cluster and cross cluster overlap with each other in Figure 1(b). If we want to monitor whether these two clusters become disjoint, then $E(s)$ would contain some composite events, which in turn consist of object deletions from the shaded area since these events can cause the clusters to separate. However, if a number of objects (denoted by bold cross) are inserted later but before any composite event in $E(s)$ occurs as shown in Figure 1(c), then these two

[1] A composite event e is a set of events $\{e_1(k_1), e_2(k_2), \ldots, e_n(k_n)\}$ where $e_1, e_2, \ldots, e_n$ are different types of primitive events. e is said to occur only if every event e_i in the set occurs at least k_i times. For example, if $e = \{insertion(20), update(15)\}$, then e is said to occur when 20 insertions and 15 updates occur. The occurrence of a composite event e is monitored by sub-triggers explained in a later section. Once e occurs, the counters used in those sub-triggers are reset.

clusters would separate only when objects within both shaded areas are deleted. $E(s)$ therefore changes with time and needs to be updated.



(a) **(b)** **(c)**

Figure 1: Effect of Events

Therefore, there are two sets of composite events we need to consider: (1) the set of composite events that can cause C_T to become true, $E(s)$; (2) the set of composite events that can cause a change to $E(s)$, call it $F(s)$. If a composite event in $E(s)$ occurs, we need to re-evaluate C_T; whereas if a composite event in $F(s)$ happens, we have to update $E(s)$. To keep $E(s)$ updated, one approach is to update $E(s)$ on each primitive event. This is appropriate in those applications where updating $E(s)$ falls out naturally from processing the event and hence little overhead will be introduced. However, if recalculating $E(s)$ requires a significant overhead, then we need to consider other approaches. An alternative is to keep $E(s)$ updated by triggering updating of $E(s)$ when a composite event in $F(s)$ occurs. This approach is preferable if, by tracking the changes to $E(s)$, the overhead of checking C_T is reduced by more than the overhead introduced by checking whether a composite event belong to $F(s)$.

In order to achieve optimal or near optimal performance, STING+ postpones the condition evaluation until the accumulated effect of some composite event (i.e., data updates) might cause either the trigger condition to become true or the composite event set $E(s)$ to evolve. Moreover, due to the fact that the effect of an event is usually local to its neighborhood. STING+ employs a hierarchical structure with associated statistical information at the various levels of the hierarchy and decomposes the user-defined trigger into a set of sub-triggers associated with cells in the hierarchy. These sub-triggers are used to monitor composite events in $E(s)$ and change accordingly when $E(s)$ evolves. The minimum accumulated amount of updates necessary to satisfy the trigger condition is maintained incrementally using statistical information associated with the hierarchy so that updates are suspended at some level in the hierarchy until such time that the cumulative effect of these updates might cause the trigger condition to become satisfied. Actions defined in the trigger will be executed automatically once the condition is satisfied. Moreover, this scheme can also be used to support incremental query processing efficiently. Due to space limitations, we only focus on trigger processing.

This paper is organized as follows. Related work is reviewed in Section 2. Section 3 and Section 4 discuss the trigger types supported by STING+ and the STING+ structure. In Section 5, algorithms for trigger evaluation are presented. Experimental results and discussions on integrating query processing with trigger evaluation are presented in Section 6 and Section 7. Finally, we draw our conclusions in Section 8.

2 Related Work

2.1 Spatial Data Mining Systems

Much work has been done in this area in recent years. [Ng94] [Zha96] [Est96] proposed algorithms to cluster large data sets. [Kno96a] [Kno96b] [Kno97] focused on extraction of *proximity relationships* between *clusters* and *features* and boundary shape matching. [Han97] presents a system prototype for spatial data mining, GeoMiner. The major features of GeoMiner include mining several kinds of knowledge rules in spatial databases, the integration of data mining and data warehousing technologies, interactive mining of multi-level rules, and integration with commercial relational databases and GIS. [Est97] [Est98b] develop a system to detect *spatial trends* and *spatial characterizations*. A spatial trend describes a regular change of one or more non-spatial attributes when moving away from a given start point, whereas spatial characterization of a set of target objects is a description of the spatial and non-spatial properties which are typical for the target objects but not for the whole database. Not only the properties of the target objects but also the properties of their neighbors are considered during this process. A neighborhood graph is employed to facilitate the mining process.

2.2 STING

STING (STatistical INformation Grid), proposed in [Wan97], is a statistical information grid-based approach to spatial data mining. A pyramid-like structure is employed (shown in Figure 2), in which the spatial area is divided recursively into rectangular cells down to certain granularity determined by the data distribution and resolution required by applications. Statistical information for each cell is calculated in a bottom-up manner and is used to answer queries. When processing a query, the hierarchical structure is examined in a top-down manner. Cells are marked as either *relevant* or *not relevant* with certain confidence level using standard statistical tests. Only children cells of *relevant* cells are examined at next level. The final result is formed as the union of qualified leaf level cells.

STING has the following advantages [Wan97]:

- It is a query-independent approach to storage structure since the statistical information exists independently of queries. This structure 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 leaf cells. Usually, $K << N$ where N is the number of objects.

- Query processing algorithms using this structure are trivial to parallelize.
- When data is updated, we do not need to recompute all information in the cell hierarchy. Instead, updates can be handled in an incremental manner.

2.3 Active Data Mining

Active data mining has emerged recently. For example, incremental algorithms have been proposed for mining dynamic databases. [Fel97] presents an incremental algorithm for mining association rule in a dynamic database, which achieves a speed-up of several orders of magnitude compared to the non-incremental algorithm with small space overhead. An algorithm for incremental clustering for mining in a data warehousing environment is proposed in [Est98a]. Updates are collected and only affected objects are re-examined during the next query evaluation. This yields significant speed-up over the non-incremental version. However, these algorithms only aim at a predefined task. User-defined triggers are not supported. [Agr95] outlines a paradigm for active data mining in temporal databases. Data is continuously mined at a desired frequency. As rules are discovered, they are added to a rulebase. Users can specify a history pattern in a trigger which is fired when such a pattern is exhibited.

3 Spatial Data Mining Triggers

STING+ supports spatial data mining triggers monitoring both spatial regions that satisfy some condition and attribute values of objects within some spatial regions. Looking ahead, STING+ employs a hierarchical structure. Space is recursively partitioned into grid cells down to a specified granularity and is organized via the inherent pyramid hierarchy. Statistical information associated with each cell is stored to facilitate the trigger evaluation. A **region** in STING+ is defined as a set of adjacent leaf level cells. In addition, object density and attribute conditions in STING+ are defined in terms of leaf level cells as well[2]. More specifically, the *density* of a leaf level cell is defined as the ratio of the number of objects in this cell divided by the area of this cell. A region is said to have a certain density c iff the density of every leaf level cell in this region is at least c. Conditions on attribute values are defined in a similar manner. Two kinds of conditions can be specified by the user. One condition is an *absolute* condition, i.e., the condition is satisfied when a certain state is reached. For instance, one can specify the condition to be "there is a region where at least 10 cellular phones are in use per squared mile with total area at least 10 squared miles". The other type of condition is a *relative* condition, i.e., the condition is satisfied when a certain degree of change has been detected. For

[2] Larger units, corresponding to some level of the pyramid closer to the root, can be used to define density and attribute condition and can be supported by the STING+ structure with minor change of the algorithm for trigger evaluation. Due to simplicity of explanation, we always assume leaf level cell is the basic unit for density and attribute conditions.

example, the condition can be specified as “when the cellular phone usage drops by 20% in the region where at least 10 cellular phones are in use per squared mile with total area at least 10 squared miles compared to now”. Therefore, four categories of triggers are supported by our system.

1. **region-trigger**: *absolute* condition on certain regions,
2. **attribute-trigger**: *absolute* condition on certain attributes,
3. **region-δ-trigger**: *relative* condition on certain regions,
4. **attribute-δ-trigger**: *relative* condition on certain attributes.

A conjunction of the above four categories of triggers can also be specified. However, due to space limitations, we do not elaborate in detail on conjunctions in this paper. Since it is trivial to decompose a conjunction trigger into several primitive triggers and then process each one using the same algorithms as for primitive triggers, we will omit the algorithms for evaluating a conjunction trigger.

STING+ provides users with the capability to specify the *active period* of a trigger, i.e., a time period during which the trigger is active. The optional SINCE and UNTIL clauses are used for this purpose (Example 3.3, 3.4). The default values are SINCE NOW and UNTIL FOREVER, respectively. Sometimes, the user wants to be notified only if the specified pattern exists for a certain amount of time. Such a condition can be specified in the VALID clause (Example 3.1).

The region(s) on which a trigger is defined can either be *fixed* through the active period or change with time. STING+ requires the conditions for *fixed* region(s) and for *variable* region(s) to be specified in different clauses since they need to be handled differently. The fixed region(s) are specified in the LOCATION clause (Example 3.1), whereas the variable regions are specified as region conditions in the WHERE clause (Example 3.2). Note that the fixed region(s) can be the result of another region query (Example 3.3). The default region is the entire space (Example 3.4).

Usually, the action in a trigger (defined in the do-clause) will be executed only once; namely the first time the condition is satisfied during the active period. The default declaration has this semantics. However, in some cases, the existence of old patterns is not as interesting as the appearance of new patterns. The user may want to be informed only when a new pattern satisfying the condition first appears. For example, a user may only want to be notified whenever a new region where cellular phones are heavily used emerges in order to re-allocate the bandwidth. In this case, the user needs to define the trigger as a REPEAT TRIGGER.

The BNF of the query language is given in Appendix A. Below are several examples, one for each category of trigger.

Example 3.1 Region-trigger: *Trigger a market study when there exists a continuous region within California for at least 120 minutes, where at least 10 cellular phones are in use per squared mile and at least 70% of the them are long distance calls (distance larger than 100 miles) with total area at least 50 squared miles with 90% confidence.*

DEFINE TRIGGER ex1
ON cellular-phone
WHEN SELECT REGION
WHERE DENSITY IN RANGE (10, ∞)
AND distance WITH PERCENT (70, 100) IN RANGE (100, ∞)
AND AREA IN RANGE (50, ∞)
AND WITH CONFIDENCE 0.9
LOCATION California
VALID 120
DO market-investigation

Example 3.2 Attribute-trigger: *Trigger a market study when the average call length is greater than 10 minutes within the region where at least 10 cellular phones are in use per squared mile and at least 70% of the them are long distance calls (distance larger than 100 miles) with total area at least 50 squared miles with 90% confidence.*

DEFINE TRIGGER ex2
ON cellular-phone
WHEN SELECT AVERAGE(length) IN RANGE (10, ∞)
WHERE DENSITY IN RANGE (10, ∞)
AND distance WITH PERCENT (70, 100) IN RANGE (100, ∞)
AND AREA IN RANGE (50, ∞)
AND WITH CONFIDENCE 0.9
DO market-investigation

Example 3.3 Attribute-δ-trigger: *Trigger bandwidth re-allocation when the average conversation length increases by 20% from now until December 31, 1998 in those region(s) where at least 10 cellular phones are in use per squared mile and all of them are long distance calls (distance larger than 100 miles) with total area at least 100 squared miles right now.*

DEFINE TRIGGER ex3

ON cellular-phone

WHEN SELECT AVERAGE(length) INCREASE PERCENT (20, ∞)

LOCATION SELECT REGION

WHERE DENSITY IN RANGE (10, ∞)

AND MIN(distance) IN RANGE (100, ∞)

AND AREA IN RANGE (100, ∞)

DO bandwidth-reallocation

UNTIL 12/31/1998

Example 3.4 Region-δ-trigger*: Trigger bandwidth reallocation when the total area occupied by those regions where at least 10 cellular phones are in use per squared mile and all of them are long distance calls (distance larger than 100 miles) with total area at least 50 squared miles increases by at least 10 squared miles from August 1, 1998 to July 31, 1999.*

DEFINE TRIGGER ex4

ON cellular-phone

WHEN SELECT SIZE(REGION) INCREASE RANGE (10, ∞)

WHERE DENSITY IN RANGE (10, ∞)

AND AREA IN RANGE (50, ∞)

AND WITH CONFIDENCE 0.9

DO bandwidth-reallocation

SINCE 8/1/1998

UNTIL 7/31/1999

4 STING+ Structure

To facilitate trigger processing in STING+, we adopt a similar hierarchical structure to that used in [Wan97]. This structure can be regarded as a summary representation of data at different levels of granularity to allow triggers to be evaluated efficiently without recourse to the individual objects. The root of the hierarchy is at level 1 and corresponds to the whole spatial area. Its children cells are at level 2, etc. A cell at level i corresponds to the union of the areas of its children at level $i+1$. The size of leaf level cells is dependent on the density of the objects. As a rule of thumb, we choose a granularity for the leaf level such that the average number of objects in each cell is in the range from several dozens to several thousands. Figure 2 illustrates the hierarchical structure in the two dimensional case.

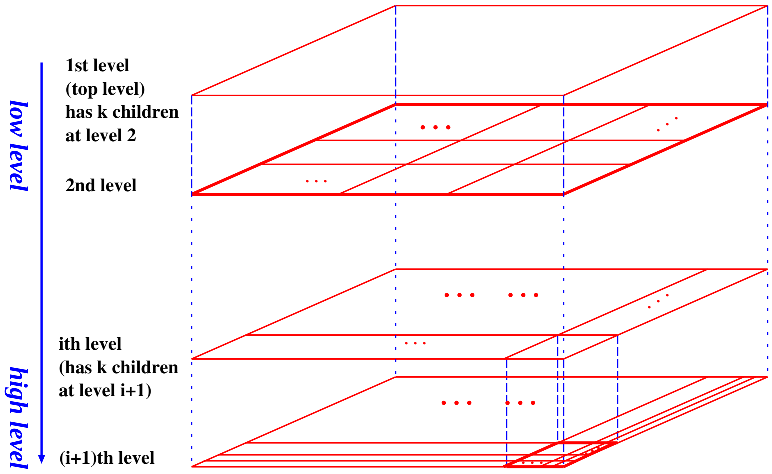


Figure 2: Hierarchical Structure

For each cell c_m at level l in a m dimensional space, its neighbors along dimension i $(1 \leq i \leq m)$ are those cells at level l whose projections on all other dimension $j (1 \leq j \leq m, j \neq i)$ coincide with that of c_m and whose projections on dimension i are *adjacent* to c_m. Note that a cell can have as many as two neighbors in each dimension or a total of $2m$ neighbors. For any region r, a leaf level cell x is an *interior boundary cell* of r iff x is within r but at least one of x's neighbor is outside r. In this paper, we assume that our space is of two dimensions unless otherwise specified. However, all results can be generalized to higher dimensional space with minor modification. Note that, in the cases where the dimensionality is very high and/or the data distribution is extremely skewed, this structure may become less efficient. In such a scenario, a more sophisticated indexing structure, such as PK-tree [Wan98b], may be employed to serve as the underlying structure[3]. For example, in Figure 3(a), C is a cell in two dimensional space. Its neighbors in dimension 1 are A and B while D and E are C's neighbors in dimension 2. A, B, D, and E are all C's neighbors. In Figure 3(b), the shaded cells are the interior boundary cells of the region whose contour is the bold line.

For each cell in the hierarchy, a set of statistical parameters are maintained. The choice of parameter may be application-dependent. In this paper, we assume the following parameters are maintained.

- attribute-independent parameter:
  - n — number of objects in this cell
- attribute-dependent parameters for *each* numerical attribute:
  - m — mean of all values of the attribute in this cell

[3] Since sibling cells may be of different size, the definitions of *cell adjacency* and *region* have to be relaxed to accommodate it. We will not discuss this issue further in this paper.

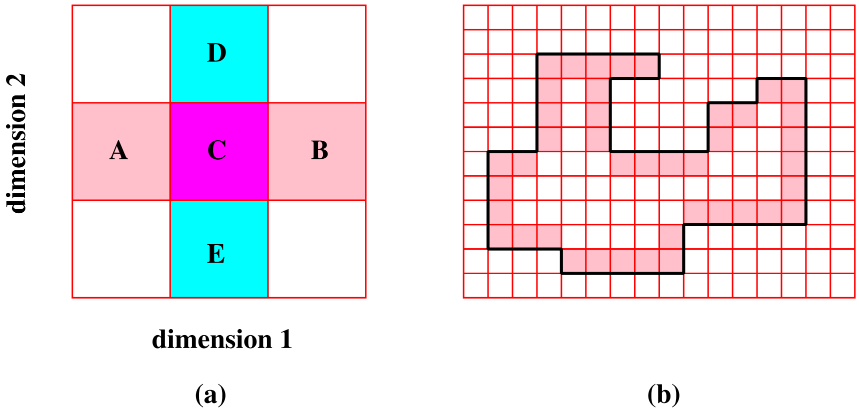


Figure 3: Neighborhood and Region Interior Boundary

– 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.

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.

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

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 not changed;
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, if the parameters of each of four lower level cells are as shown in Table 1, then the parameters of the current cell would 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 reason: 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 that $confl = 10$. So, $dist$ is kept as NORMAL ($\frac{confl}{n} = 0.045 < 0.05$).

Note that 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. Once the structure has been generated, it can be maintained incrementally to accommodate new updates.

In STING+, user-defined triggers are decomposed into sub-triggers associated with individual cells in order to benefit from the statistical information during trigger evaluation. These sub-triggers monitor composite events in $E(s)$

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

as well as the change of $E(s)$. When a composite event in $E(s)$ occurs, the trigger condition C_T will be checked. At the same time, if $E(s)$ changes, some sub-triggers might be removed or added accordingly. As we will see later in this section, when a region evolves, sub-triggers on this region are modified accordingly. Two pairs of *sub-triggers* can be set on a cell to control whether or not insertion/deletion/updates need to be forwarded to higher levels in the hierarchy by monitoring density and attribute conditions, respectively. These sub-triggers have parameters that are calculated when they are activated; and these will be described shortly. Sub-triggers set on leaf level cells also keep track of changes to region(s) satisfying the condition in C_T whereas intermediate level sub-triggers only account for propagation of updates to higher levels when some conditions are met.

Insertion-sub-triggers and *deletion-sub-triggers* are sub-trigger types (referred to as *density-sub-triggers*) used to monitor density changes of a cell, i.e., whether the number of objects in this cell reaches a threshold n_t from below or above, respectively. The minimum number of insertions (or deletions) needed to make the trigger condition true, referred to as n_{ins} (or n_{del}), is stored as a parameter. It is obvious that only object insertion/deletion[4] can affect the condition of these two sub-triggers; a change of attribute value has no effect on object density. Another pair of sub-triggers (referred to as *attribute-sub-triggers*) are *inside-sub-trigger* and *outside-sub-trigger*. They monitor whether an aggregate (such as MIN, MAX, AVERAGE, etc.) or a certain percentage (e.g., 80%) of attribute values enters or leaves a range $[r_l, r_u]$, respectively. The accumulated amount of updates to this attribute necessary to achieve this goal, $\sum \delta attr$, is stored as a parameter. Different parameters are used for different types of aggregates. Object insertions/deletions and updates on attribute values affect these two sub-triggers. Upon satisfaction of sub-trigger conditions, the actions taken by the inside-sub-trigger and the outside-sub-trigger are the same as those taken by the insertion-sub-trigger and deletion-sub-trigger, respectively. The main difference between density-sub-triggers and attribute-sub-triggers is that the associated parameters are different: one focuses on number of objects and the other accounts for attribute values as well. Therefore, we only explain the management of density-sub-triggers in detail.

[4] Update that changes spatial location can be viewed as a deletion followed by an insertion.

Density-sub-triggers can be used to monitor both "dense" region (i.e., the density is above some threshold) and "sparse" region (i.e., the density is below some threshold). Since the procedures for these two cases are analogous in spirit, we only elaborate the procedure to monitor a dense region in this section.

Density-sub-triggers can be set on cells at any level except the root. We first discuss the leaf level density-sub-triggers. Given region(s) R with density at least c, the leaf level density-sub-triggers are used to monitor any change to the area of R. Events that could affect R are deleting objects from R and inserting objects into the neighborhood area of R. Note that inserting objects in R or deleting objects from a location outside R does not change R. In turn, deletion-sub-triggers are always set on those leaf level cells within R whereas insertion-sub-triggers are always set on those cells which are outside but adjacent to R. Parameters for these sub-triggers are calculated from the statistical information associated with the leaf cells. They are $n_{del} = c \times a_b - n$ and $n_{ins} = n - c \times a_b$ respectively, where a_b is the area of a leaf level cell. Once the condition of a deletion-sub-trigger is satisfied on a leaf level cell (i.e., the density of this cell goes below c), the following procedure is executed.

1. Replace the deletion-sub-trigger with an insertion-sub-trigger on this cell.

2. If any neighbor cell, x, which has an insertion-sub-trigger is no longer adjacent to R, remove the insertion-sub-trigger on x.

3. Adjust the area of the involved region. (If a region splits, the area of each subregion has to be recalculated.)

On the other hand, actions triggered by an insertion-sub-trigger involves the following steps.

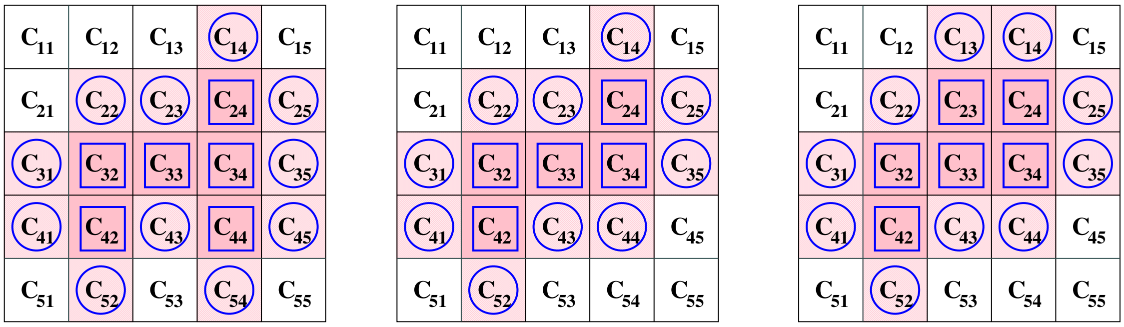
1. Replace the insertion-sub-trigger with an deletion-sub-trigger on this cell.

2. Adjust the area of the involved region(s).

3. If any of its neighbors do not have a density-sub-trigger yet, set an insertion-sub-trigger on that neighbor.

Note that the sub-trigger placed on a cell may vary over time. An example is shown in Figure 4. C_{24}, C_{32}, C_{33}, C_{34}, C_{42}, and C_{44} are cells whose densities are at least c. These cells are shown in dark shading in the figure. Deletion-sub-triggers are placed on these cells. The surrounding cells in light shade are those cells that are outside the region (i.e., whose density is below c) but adjacent to at least one interior boundary cell. Insertion-sub-triggers are placed on these cells. This is illustrated in Figure 4(a). Figure 4(b) and (c) show the scenario that C_{44}'s density goes below c, then C_{23}'s density goes above c, respectively.

It is obvious that at any time the number of deletion-sub-triggers on leaf level is the number of leaf level cells within R, $N(R)$, and the number of leaf level insertion-sub-triggers is between $NB(R) + 4$ (Figure 5(a)) and $2 \times NB(R) + 2$ (Figure 5(b)) in a two dimensional space, where $NB(R)$ is the number of interior boundary cells



(a) Insertion-sub-triggers and deletion-sub-triggers are set.

Insertion-sub-trigger

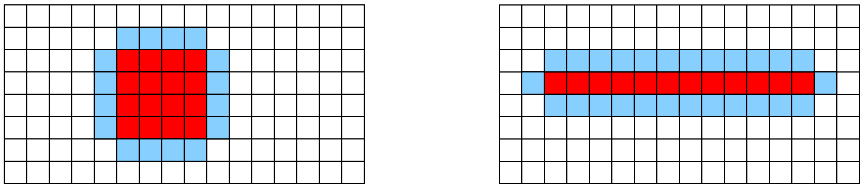
Deletion-sub-trigger

(b) The deletion-sub-trigger on C_{44} is triggered. As a result,
- this deletion-sub-trigger is replaced by a insertion-sub-trigger;
- The insertion-sub-triggers on C_{45} and C_{54} are removed;
- The area of this region is decreased by 1.

(c) The insertion-sub-trigger on C_{23} is triggered. As a result,
- this insertion-sub-trigger is replaced by a deletion-sub-trigger;
- an insertion-sub-trigger is placed on C_{13};
- the area of this region is increased by 1.

Figure 4: Leaf Level Insertion and Deletion Sub-triggers

of R. Moreover, since $4 \times \lceil\sqrt{N(R)}\rceil - 4 \leq NB(R) \leq N(R)$, the total number of density-sub-triggers set on R is between $N(R) + 4 \times \lceil\sqrt{N(R)}\rceil$ (Figure 5(a)) and $3 \times N(R) + 2$ (Figure 5(b)).



(a) N(R) = 16, NB(R) = 12;
There are 16 leaf level deletion-sub-triggers and 16 leaf level insertion-sub-triggers.

(b) N(R) = 12, NB(R) = 12;
There are 12 leaf level deletion-sub-triggers and 26 leaf level insertion-sub-triggers.

Deletion-sub-trigger **Insertion-sub-trigger**

Figure 5: Number of Sub-triggers

For each intermediate level cell, an insertion-sub-trigger (deletion-sub-trigger) is set iff one of its child cells has an insertion-sub-trigger (deletion-sub-trigger). The value of the parameters n_{ins} and n_{del} are set to be the minimum of that of its children cells. The action of a sub-trigger on an intermediate level cell is to propagate the update held at this cell to its children at next higher level and then recalculate the sub-trigger parameters. An insertion- (deletion-) sub-trigger is removed if all insertion- (deletion-) sub-triggers on its children are removed.

Moreover, STING+ allows two additional types of sub-triggers *expand-sub-trigger* and *shrink-sub-trigger* to be set *only* on leaf level cells to track expansion and shrinking of region(s) Q from a given time, t_1, respectively. Two variables A_{shr} and A_{exp}, which are updated by the shrink-sub-triggers and expand-sub-triggers respectively, are used to keep track of the area of $Q - Q'$ (shrunken part) and $Q' - Q$ (expanded part), where Q' is the region(s) that evolved from Q. The initial value of A_{shr} and A_{exp} are zero at time t_1. *Shrink-sub-triggers* are set on leaf level cells within Q at time t_1 since these cells have the potential to leave Q and hence cause Q to shrink. Once the shrink-sub-triggers are set, they will never be removed or replaced by expand-sub-triggers during the active period. Also, no new shrink-sub-trigger will be activated on other cells after t_1. This enables STING+ to track the shrinkage of an evolving region Q'(at the current time) against the original region Q (at time t_1). *Expand-sub-triggers* are used to monitor the expansion of Q' against Q. Therefore, these expand-sub-triggers are set on cells which are outside the original region(s) Q but which have the potential to join the region and cause it to expand. At time t_1, expand-sub-triggers are placed on those cells outside but adjacent to Q. Unlike shrink-sub-triggers, expand-sub-triggers can be set and removed dynamically.

When a leaf level cell y with a shrink-sub-trigger leaves the region(s), the value of A_{shr} is increased. Expand-sub-triggers on y's neighbors which are not adjacent to the region any longer are removed. However, y still keeps the shrink-sub-trigger. Later, if y again joins the region, the value of A_{shr} is decreased and expand-sub-triggers are set on its neighbors which do not have shrink- or expand-sub-triggers. When a leaf level cell x with an expand-sub-trigger joins the region, the value of A_{exp} is increased and expand-sub-triggers are set on all its neighbors that do not have shrink/expand-sub-triggers. In the case that a leaf level cell with expand-sub-trigger leaves the region again, the value of A_{exp} is decreased and the expand-sub-triggers on its neighbors which have expand-sub-trigger but are not adjacent to the region any longer will be removed.

An example is shown in Figure 6. The solid bold line contours the region Q. Shrink-sub-triggers and expand-sub-triggers are represented by dark shade and light shade on a cell respectively.

Since more than one trigger defined by users may co-exist at the same time, there may be more than one sub-trigger on a cell. All insertion/deletion/updates forwarded by its parent cell are collected by this cell until one of the sub-triggers set on this cell becomes valid or in an extreme case there is no space to hold more suspended updates. From our experiments, the storage overhead of sub-triggers in STING+ is 10% compared to STING on average.

5 Trigger Evaluation

In order to handle triggers efficiently, we evaluate the trigger condition incrementally when an update occurs. Triggers are decomposed into *sub-triggers* associated with cells in the hierarchy. Updates to the database will not be forwarded to higher levels in the hierarchy until certain conditions are met. This is based on the observation that the effect of a single update (or small number of updates) usually is not enough to make the trigger condition valid. It is therefore not necessary to apply the update immediately and re-evaluate the trigger condition every time an update occurs.

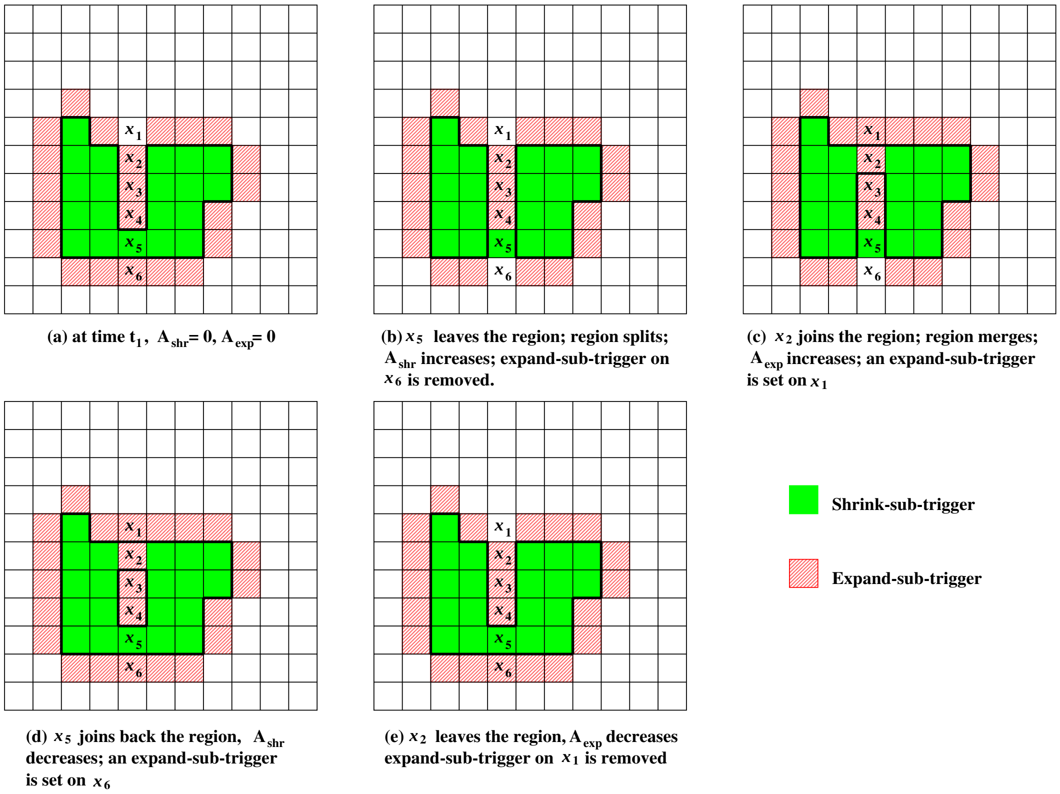


Figure 6: Shrink and Expand Sub-triggers

If the database is not used to serve other applications, it is more efficient to postpone updates to the database[5] and also evaluations of trigger conditions until it is possible that the accumulative effect of updates collected can make the trigger condition become true. Sub-triggers in STING+ are used for this purpose. The associated parameters are calculated based on statistical information from the current database and the condition specified in triggers. This enables the system to determine whether a trigger condition can transition to true by examining only a few cells.

The usually complicated condition specified in a user-defined trigger makes trigger decomposition crucial to efficient evaluation. STING+ employs a "step-by-step" strategy based on the observation that a trigger condition is a conjunction of predicates $P_1 \cap P_2 \cap \ldots \cap P_n$ and can not become true if one predicate is false[6]. It is therefore not necessary to evaluate all predicates at the same time. Instead, predicates can be evaluated in a certain order: the ith predicate is tested only when all previous $i-1$ predicates are true. Moreover, costs for evaluating different predicates may be different, the order should be chosen in such a way that the total cost of evaluation is minimum. That is, P_1, P_2, ..., P_n should be evaluated

[5] Applying updates as a batch will save disk I/O if the structure is stored on disk.

[6] Triggers with disjunctive conditions can always be rewritten to an equivalent set of triggers with only conjunctive conditions.

in the order $P_{k_1}, P_{k_2}, \ldots, P_{k_n}$ such that $cost(P_{k_1}) \times t_1 + cost(P_{k_2}|P_{k_1}) \times t_2 + \ldots + cost(P_{k_n}|P_{k_1}P_{k_2}\ldots P_{k_{n-1}}) \times t_n$ is minimized, where k_1, k_2, ..., k_n is a permutation of $1, 2, \ldots, n$, and $cost(X)$, $cost(Y|X)$ are the average cost to evaluate predicate X and that of Y given X is true, respectively, and t_i is the number of times P_{k_i} needs to be evaluated. In particular, the trigger evaluation process in STING+ employs the order {location, density condition, attribute condition,...} and is therefore divided into phases, one for each predicate. The reason for choosing this order is that the location only needs to be evaluated once and the cost is at most the cost to answer a region query in STING. It can be regarded as constant in the trigger evaluation cost function. Moreover, if the location is fixed, unnecessary sub-triggers set on cells outside the location can be avoided and hence save the evaluation cost of other predicates. Therefore, STING+ always computes the location first. Evaluation of both the density condition and the attribute condition involves activation of sub-triggers on cells. Looking ahead, sub-triggers set during an earlier phases will exist longer than those set in a later phase. Thus, it is better to first evaluate the predicate that takes less time to handle. It is shown in a later section that attribute-sub-triggers are more costly than density-sub-triggers. Therefore, STING+ evaluates the density condition before the attribute condition.

The first three phases are common for all four types of triggers. The fourth phase is used to account for the SELECT clause and hence is different for different trigger types.

Given the following trigger,

DEFINE TRIGGER trigger-name
ON map
WHENSELECT <select-clause>
    WHERE DENSITY IN RANGE (c, ∞)
    AND <attr-func> IN RANGE (a, b)
    AND AREA IN RANGE (A, ∞)
    AND WITH CONFIDENCE $1 - \alpha$
    LOCATION <location>
DO alarm
SINCE $time_1$
UNTIL $time_2$

STING+ evaluates the LOCATION clause in the **first phase**. <location> specifies the fixed region(s) on which the trigger is defined. There are two cases: the first case is that <location> is a list of names and/or polygons whereas in the second case the <location> is another region query. In the second case, we need to find the region by processing this region query in the same manner as done in STING [Wan97].

Once we determine the spatial area S over which the trigger is defined, the evaluation process enters the **second phase** where the DENSITY clause is evaluated. The evaluation of attribute conditions is delayed until the DENSITY condition is satisfied. For each leaf level cell in S, we calculate the number of objects which need to be inserted in order to achieve the required density in this cell, which is $\max\{0, c \times a_b - n\}$. Then, we calculate the minimum number of objects t needed overall in order to have $\lceil A/a_b \rceil$ leaf level cells with density at least c[7]. The *density-trigger* is set to monitor whether there is a region satisfying both the DENSITY and the AREA conditions. A density-trigger is set on the root in such a way that updates to higher levels are postponed until at least t insertions have been submitted. Immediately after applying a set of t insertions, the value of t is recalculated. This process continues until t is below a threshold t_0. Then, *density-sub-triggers* (i.e., insertion-sub-triggers and deletion-sub-triggers) are set on those regions with density c and area at least A_t to monitor the area change, where $A_t = A/t_1$ where t_1 is a threshold set by the user or the system. All the remaining space is still accounted for by the density-trigger at the root. Note that the large regions and small regions are handled differently.

The above procedure continues until there is at least one region with density c and area at least A. Let R be the set of such qualified regions. The trigger process then enters the third phase on R. Note that the remaining space is still in the second phase.

The **third phase** is analogous to the second phase except that the attribute $attr$ is considered instead of the density. In this phase, the accumulated amount of updates (on $attr$ value) necessary to satisfy the attribute condition is calculated for each leaf level cell within R. For example, if the <attr-func> defined by the user is AVERAGE($attr$), then the accumulated amount of updates u is $(a-m) \times n + a \times \delta n \le u \le (b-m) \times n + b \times \delta n$, where δn is the change in the number of objects. Then we calculate the total accumulated amount of updates on $attr$, $t_u(attr) = \sum \delta attr$ needed in order to have $\lceil A/a_b \rceil$ cells within R satisfy the attribute condition. An $aggregate_{attr}$*-trigger* is set at the root to suspend all updates until $t_u(attr)$ amount of updates on $attr$ occurs. $t_u(attr)$ is recalculated every time when all updates are forwarded to leaf level. This procedure continues until $t_u(attr)$ is below a threshold. Attribute-sub-triggers are set on those regions satisfying the attribute condition and with area at least A_t within R to monitor the area change until there is a region satisfying both attribute condition with area at least A within R. Such qualified region(s), referred to as Q, will enter the fourth phase.

Note that during this phase, all sub-triggers of the previous phase are still active. If the density condition is violated again after a region enters the third phase, all attribute-sub-triggers placed on this region are suspended until this region becomes qualified again.

The **fourth phase** accounts for the SELECT clause and hence is different for each type of trigger.

[7] A heap is used to calculate t.

5.1 Region-trigger

For region-triggers, no additional requirement is specified in the SELECT clause. Therefore, this phase is omitted when processing a region-trigger.

5.2 Attribute-trigger

Since the attribute-trigger is used to detect certain attribute patterns, the SELECT clause is usually defined as "SELECT $<attr2$-func$>$ IN RANGE (e, f)". The minimum accumulated amount of updates overall on $attr2$, $t_u(attr2)$, needed to cause the condition to become true is calculated in the same manner as in the third phase. An $aggregate_{attr2}$-$trigger$ is set at the root to suspend all updates until $t_u(attr2)$ amount of updates on $attr2$ occurs. $t_u(attr2)$ is recalculated every time updates are applied to higher levels. The procedure continues until $t_u(attr2)$ reaches zero.

5.3 Region-δ-trigger

There are two common ways to quantify a region change. One is to use size difference and the other is based on boundary distance [Hor97]. STING+ supports both and enables users to choose their favorite measure. If SIZE(REGION) is used, the region change will be measured using area of the symmetric difference between the original region and the current region. For example, in Figure 7(a), the original region is denoted by a solid contour whereas the current region is denoted by dashed contour. The symmetric difference between these two regions is shown as the shaded region whose area is used as a measure of change.

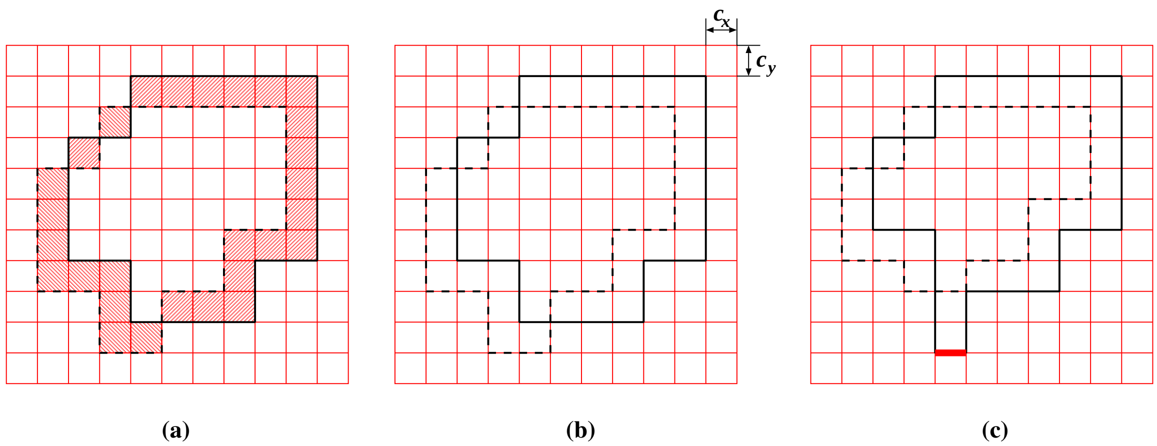


Figure 7: Size Difference and Boundary Distance of Region

Another measure of region difference is the boundary distance, which is defined as the smallest number dis such that every point on the boundary of the evolved region is within a distance dis of some point on the boundary of the original region and vice versa. In Figure 7(b)(c), assume that the size of each leaf level cell is $c_x \times c_y$ and Euclidean distance is used. Then the boundary distance in Figure 7(b) is $\sqrt{c_x^2 + c_y^2}$. However, the boundary distance in Figure 7(c) is $2 \times c_y$ because points on some portion of the boundary of the original region (denoted by bold line in the Figure 7(c)) are at least $2 \times c_y$ distance away from any point on the boundary of the evolved region even though every point on the boundary of evolved region is within $\sqrt{c_x^2 + c_y^2}$ distance of some point on the boundary of the original region.

Suppose that the SELECT clause is specified as "SELECT [SIZE | BOUNDARY](REGION) CHANGE RANGE (e, f)" in the trigger. Shrink-sub-triggers and expand-sub-triggers are set on Q to monitor the evolution of Q via either the symmetric difference or the boundary distance until $time_2$ or $A_{shr} + A_{exp}$ reaches e.

Besides the above two measures, STING+ also supports pure area change without concern for the region movement. In this case we only take into account the area difference of the original region and the region to which it evolves. In this case, the change could be either positive or negative depending on whether the region expands or shrinks in size.

5.4 Attribute-δ-trigger

Each attribute-δ-trigger can always be rewritten as equivalent attribute triggers. For example,

SELECT AVERAGE($attr$) $\{$INCREASE | DECREASE$\}$ [RANGE | PERCENT] (e, f)

is equivalent to

SELECT AVERAGE($attr$) [RANGE $(AVERAGE_{time_1}(attr) + e, AVERAGE_{time_1}(attr) + f$
| PERCENT $(AVERAGE_{time_1}(attr) \times (1 + e/100), AVERAGE_{time_1}(attr) \times (1 + f/100))$]

where $AVERAGE_{time_1}(attr)$ is the average age at $time_1$. When an attribute-δ-trigger is submitted, it is first translated into equivalent attribute trigger and then this attribute trigger is processed by STING+ as usual. If the keyword CHANGE is used instead of INCREASE or DECREASE, the attribute-δ-trigger is equivalent to two attribute-δ-triggers: one for INCREASE and one for DECREASE. So, it is transformed into two attribute triggers and processed by STING+ simultaneously. Therefore, the fourth phase of attribute-δ-trigger is the same as that of an attribute-trigger.

6 Experimental Results

We implemented a prototype of STING+ in C and all experiments were performed on a SPARC10 workstation with SUNOS 5.5 operating system and 208 MB main memory. In this section, we will analyze three aspects of the performance of STING+: the average number of CPU cycles for handling a sub-trigger, the average number of sub-triggers for a region, and the average update costs in STING+.

In all performance measurements, we have 200,000 synthetically generated 2-dimensional data points. There are three kinds of data location distributions, i.e., uniform distribution, normal distribution with small variance, and normal distribution with large variance. There are also three kinds of attribute value distributions: uniform, normal with small variance, and normal with large variance. For each combination of location and attribute value distribution, we generate two sets of data, each with a different random number generator. Therefore, we have eighteen sets of data. We chose twelve triggers (three in each category) to test the performance. We insert a set of 10,000 randomly selected data points with different attribute values, then delete 10,000 randomly selected data points, and update the attribute values of another 10,000 data points. In these experiments, there are seven levels in STING+, and the bottom level consists of 4096 cells.

Since STING+ suspends updates and stores the parameters of each sub-trigger, there is some storage overhead. After generating STING+ structures, we found that the average size of STING+ structure is 0.75MB and overhead of STING+ is about 10% compared to STING.

6.1 Cost for Handling a Sub-Trigger

There are six types of sub-triggers, and the costs for handling them are different. Therefore, we will analyze them separately. Table 2 shows the cost of handling a sub-trigger. Since the only action for an intermediate level sub-trigger is forwarding the update down or suspending it, the CPU cycles consumed for this function is minimal. In addition, the actions taken by all four types of sub-triggers are the same, thus, the costs for each of intermediate level sub-triggers are the same. (Expand- and shrink- sub-triggers are set only on the leaf level.)

However, the leaf level sub-triggers are much more expensive (excluding expand-sub-trigger and shrink-sub-trigger) as in Table 2 because it has to remove or set other sub-triggers and recalculate the parameters. Furthermore, the costs for the different types of sub-triggers are significantly different. It is more expensive to handle a leaf level insertion-sub-trigger than a leaf level deletion-sub-trigger because new leaf level sub-triggers on neighboring cells are created when an insertion-sub-trigger is triggered, while no new sub-triggers will be created on the neighboring cells when a leaf level deletion-sub-trigger is triggered. The cost of creating a new sub-trigger is much larger than that of removing a sub-trigger. For the same reason, handling a leaf level inside-sub-trigger is much more expensive than handling an leaf level outside-sub-trigger. Furthermore, the attribute-sub-triggers are more expensive than density-sub-triggers because calculating the parameters of attribute-sub-triggers is more expensive than that of density-sub-triggers. Expand- and shrink-sub-triggers are much simpler than attribute-sub-triggers and density-sub-triggers, and therefore take much less time to handle.

During an update, the overall number of intermediate level sub-triggers involved is at most the height of the STING+ pyramid. And it remains more or less constant. The largest cost is the evaluation of leaf level sub-triggers. However, for each update, at most one leaf level sub-trigger is evaluated. Therefore, the overall cost of handling sub-triggers in an update is small.

	insertion-	deletion-	inside-	outside-	expand-	shrink-
Intermediate Level Sub-triggers	3812	3803	3789	3807	N/A	N/A
Leaf Level Sub-triggers	8055	5775	11212	8164	2126	2087

Table 2: Average CPU cycles for handling each type of sub-trigger

6.2 Average Number of Leaf Level Sub-Triggers for a Region

The number of sub-triggers created for a trigger is not only dependent on the number of regions and the size of regions but is also related to the shape of regions as well. Since the cost of handling a leaf level sub-trigger is much higher than that of an intermediate level sub-trigger, we focus on the average number of leaf level sub-triggers created for a given region. For a given trigger, the number of significantly large regions is highly dependent on the data and the trigger condition so that we will not focus on this aspect.

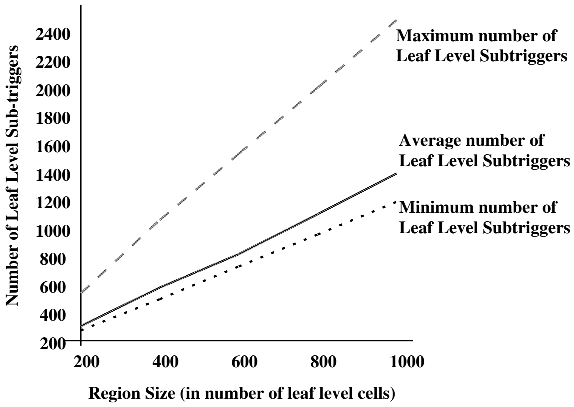


Figure 8: Number of Leaf Level Sub-triggers for a Given Size Region

For regions that have similar area and shape we found that the number of leaf level sub-triggers used for different categories of triggers is similar. As a result, we do not distinguish among the four categories of triggers. The number of large regions varies significantly from one trigger to another and from one data set to another. The total number of sub-triggers for all these tests is about three thousand. We group regions according to their size. Figure 8 illustrates the number of leaf level sub-triggers placed on a given size region. The top line, middle line, and bottom line indicate the maximum number, average number, and minimum of leaf level sub-triggers for a given range of region size. The variation of the number of sub-triggers for a given size region is due to different shapes of the regions. We provided the theoretical bounds on the number of leaf level sub-triggers. From Figure 8, we can see that the average number of sub-triggers is much closer to the minimum number because it is very rare that a region will have the shape that yields the maximum number of sub-triggers (e.g., a flat shape (Figure 5(a))).

6.3 Update Cost

Cost varies for each update. If at the time the root is accessed, it can be determined that an update can not possibly make the trigger condition valid, then the update will be suspended at the root. The cost of this type of update is very minimal. However, when an update comes in to the root and it is deemed that the condition in the trigger may be satisfied, all suspended updates will be applied to a higher level. Then this type of update may be very expensive. We compare the average update cost of STING+ to the average update cost of STING. The result is that STING+ only consumes slightly more CPU cycles than STING on average for an update. Furthermore, since STING+ can bundle updates and apply them to the database as a batch, it can potentially reduce the number of I/O's.

When an update comes in, STING has to apply it into the leaf level cells and update all parameters for all relative cells. However, STING+ is much more flexible and does not apply updates unless it is deemed that this update may cause the trigger condition to become true. In addition, if the resources become available, e.g., CPU becomes idle, STING+ then can apply all suspend updates to the bottom level cells and make proper changes for all relative parameters. This can reduce the effective cost of updates by avoiding consuming CPU cycles and I/O bandwidth when the system is highly loaded. We found from experiments that the average overhead incurred by trigger maintenance is between 10% and 15%.

As mentioned before, one alternative to STING+ is to query the data periodically until a pattern appears. One natural question to ask is what is the maximum frequency of to submitting queries to STING so that it consumes less CPU cycles than STING+ overall. We choose the number of updates (include insertions, deletions, and attribute updates) as the measurement of period. Then the break-even point of a period should be $\frac{Q_{sting}}{U_{sting+}-U_{sting}}$ where Q_{sting} is the average cost of answering a query in STING, and U_{sting+} and U_{sting} are the average cost of an update in STING+ and STING, respectively. From our experiments, it is only profitable for STING if the period is set to be larger than 4000 updates. However, this could raise problems such as severe delay and missing of the pattern which appears and disappears quickly. Note that STING+ can produce the result without any delay. In order to achieve the same goal by periodic query re-submission on STING, the query condition has to be checked for each update. In such a case, STING+ achieves three orders of magnitude improvement over it.

7 Discussion: Integration of Query Processing with Trigger Evaluation

In previous sections, we focused on efficient evaluation of trigger condition in a spatial database environment using the sub-trigger technique. Once the condition is satisfied, the action defined in the trigger is executed. This action might be a pure system procedure call or a human interactive process. In many cases, a spatial data mining query is specified as the trigger action, which may require a significant amount of computing resources. By doing so, the user intends to acquire knowledge related to certain aspects of the database once some given event occurs. In many real-time applications, the result of such query may be required to be returned to the user immediately or within some short period. As a result, performing query process upon satisfaction of trigger condition would not be able to meet such a deadline. In this case, some degree of precomputation of the query (before the trigger condition is fulfilled) becomes necessary.

For the simplicity of explanation, we assume that the entire system dedicates to the trigger processing and at most one trigger is processed at any time[8]. Assume that it takes t_Q time to process a query Q on the entire database but the result of the query has to be returned within time t_R upon the satisfaction of the trigger condition, where $t_Q >> t_R$. In order to meet this deadline, we have to begin the query evaluation before the trigger condition becomes true. If the size of the database does not change significantly over time, we can regard t_Q as a constant number. The spatial locality property (i.e., the effect of an update is usually local to its neighborhood) provides the foundation for achieving such a goal. This property indicates that an obsolete result of a query is still "valid" *partially* if only a small portion of the data updates. For a given query Q and a database D, the query result R is essentially a function of Q and D_T where D_T is the database status at time T. In the remainder of this section, we use R_T to denote the query result corresponding to the database status at time T. Let π be the average time to integrate a new update with an obsolete query result. If an obsolete result R_{T_1} exists, then at a later time T_2, unless there are more than $\frac{t_Q}{\pi}$ new updates, it would be more efficient to use R_{T_1} as the basis to generate the new result R_{T_2} than to process the query from sketch.

Based on this observation, we devise the following twofold algorithm:

1. Before the trigger condition is satisfied (say, at time T_Q), process the query on the entire database to obtain the result R_{T_Q}[9].

2. Once the trigger condition becomes satisfied (say, at time T_s), process updates posted from time T_Q to time T_s and combine with R_{T_Q} to generate the query result R_{T_s}[10].

---

[8] Interested readers please refer to [Wan98] for analysis of more complicated scenarios.

[9] Note that R_{T_Q} will become obsolete if some new update occurs after T_Q.

[10] Note that this step has to be done within time t_R.

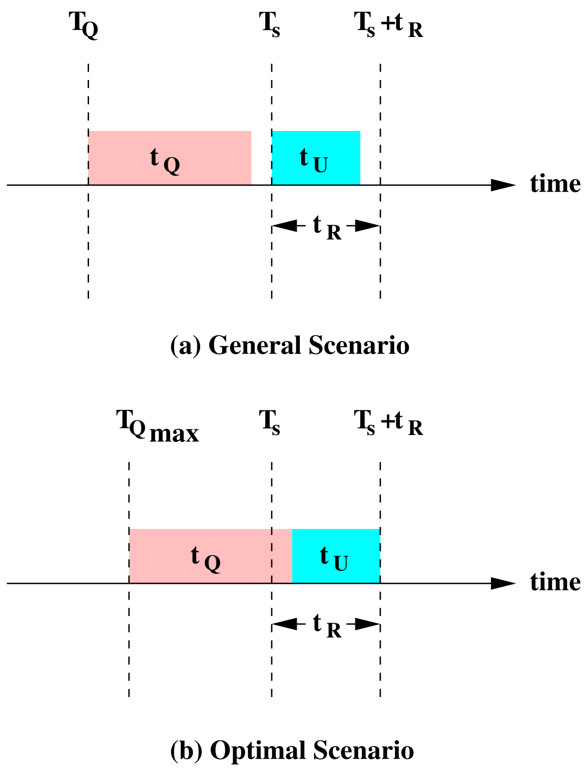


Figure 9: Time Topology for Query Processing

This is illustrated in Figure 9(a). The first step takes t_Q time to finish whereas the time consumed by the second step, referred to as t_U, depends on the number of updates occur within the time interval from T_Q to T_s. Assume that updates occur at a rate λ. Then, the number of updates posted between T_Q and T_s, referred to as N_U, is

$$N_U = (T_s - T_Q) \times \lambda.$$

We can see that the earlier we execute the query, the more updates we need to accommodate in the second step. Therefore, as far as we can still guarantee to return R_{T_s} by the time $T_s + t_R$, we prefer to execute the query as late as possible in order to minimize the computing resources consumed. In other words, we want to maximize T_Q such that

$$T_Q + t_Q + N_U \times \pi \leq T_s + t_R$$

still holds. We have

$$T_{Q_{max}} = T_s - \frac{t_Q - t_R}{1 - \lambda \times \pi}$$

which is the optimal value of T_Q (Figure 9(b)).

However, since in most cases we are unable to obtain the exact value of T_s ahead of time, it is very difficult to compute the optimal value of T_Q. We propose a heuristic to estimate $T_{Q_{max}}$. Since the sub-trigger technique is used to evaluate the trigger condition, the minimum number of updates (referred to as N_{min}) to make the trigger condition true can be obtained at any time as a byproduct. It is obvious that, at any time T, We can estimate the earliest time the trigger condition can become true as

$$T_{s_{min}} = T + \frac{N_{min}}{\lambda}.$$

Then,

$$T_{Q_{max}} \geq T + \frac{N_{min}}{\lambda} - \frac{t_Q - t_R}{1 - \lambda \times \pi}. \tag{1}$$

As time elapses (i.e., T increases), N_{min} changes as well[11]. Our objective is to determine whether the current time T is the estimated $T_{Q_{max}}$, referred to as $\widehat{T_{Q_{max}}}$. Obviously, we want $\widehat{T_{Q_{max}}}$ to be the largest possible T ($T \leq T_{Q_{max}}$) which still satisfies the above inequality. Otherwise, there would be a chance that we might miss the deadline. In fact, $\widehat{T_{Q_{max}}}$ is the time when both following conditions are true.

1. Inequality 1 still holds.

2. A future update might cause N_{min} to decrease, which in turn, would violate Inequality 1.

Therefore, we have

$$\frac{t_Q - t_R}{1 - \lambda \times \pi} \times \lambda \leq N_{min} < \frac{t_Q - t_R}{1 - \lambda \times \pi} \times \lambda + 1.$$

Once N_{min} falls into this interval, we begin to process the query. Intuitively, when N_{min} remains greater than or equal to $\frac{t_Q - t_R}{1-\lambda\times\pi} \times \lambda + 1$, we don't have to start the query processing. However, if $N_{min} < \frac{t_Q - t_R}{1-\lambda\times\pi} \times \lambda$, it is possible that we would not be able to obtain the result in time.

Because we use N_{min} and $T_{s_{min}}$ during the estimation, it might be the case that $N_U > N_{min}$ and hence $T_s > T_{s_{min}}$. This happens when some updates (occur after T_Q) do not contribute to the fulfillment of the trigger condition. Indeed, N_U might be very large in some cases. Note that we can process at most $\frac{t_R}{\pi}$ updates after the trigger condition becomes true. If N_U exceeds this limit, we will not be able to generate R_{T_s} in time. To avoid such a scenario, we have to guarantee that at most $\frac{t_R}{\pi}$ updates need to be examined to compute R_{T_s} at time T_s. A naive solution is that, after the initial query result R_{T_Q} is generated, we upgrade the previous result to an up-to-date result $R_{T'_Q}$ for every $\frac{t_R}{\pi}$ new updates, where T'_Q is the time every $\frac{t_R}{\pi}$th update occurs. So, at any time from T_Q to T_s, the number of outstanding updates[12] is at most $\frac{t_R}{\pi}$. This provides a sufficient (but not necessary) condition for our requirement. An alternative

[11] Note that the value of N_{min} can either increase or decrease as a result of the occurrence of some update along the time.

[12] An outstanding update is an update posted after the previous result was generated.

solution would be to estimate the time we need to generate an up-to-date result to reflect recent updates via a similar strategy as we estimate T_Q. Let N_o and $T'_{Q_{max}}$ be the number of outstanding updates cumulated so far and the time we have to generate the new result, respectively. Let t'_Q be the time consumed to perform this procedure. Note that we can choose to either process the entire database from sketch (which requires t_Q time) or update the previous result to reflect recent data change (which requires $N_o \times \pi$ time), depending on which one is more efficient. That is, $t'_Q = \min\{N_o \times \pi, t_Q\}$. Then, at any time T ($T > T_Q$), we have

$$T'_{Q_{max}} \geq T + \frac{N_{min}}{\lambda} - \frac{\min\{N_o \times \pi, t_Q\} - t_R}{1 - \lambda \times \pi} \tag{2}$$

and we want to determine whether T is an optimal estimation of $T'_{Q_{max}}$. By a similar analysis presented before, we obtain that every time N_o increases to

$$\frac{\min\{N_o \times \pi, t_Q\} - t_R}{1 - \lambda \times \pi} \times \lambda \leq N_o < \frac{\min\{(N_o + 1) \times \pi, t_Q\} - t_R}{1 - \lambda \times \pi} \times \lambda + 1,$$

we need to upgrade the previous result to an up-to-date version. This process continues until the trigger condition is satisfied. Figure 10 illustrates a general scenario of this process.

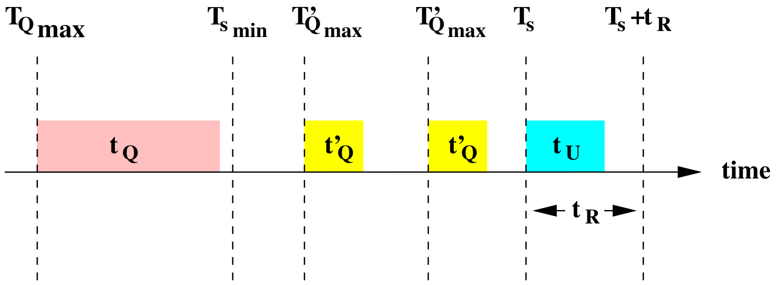


Figure 10: Heuristic Strategy

8 Conclusion

In this paper, we proposed a new approach to active spatial data mining, called STING+. In STING+, users can define triggers to monitor the change of spatial data (also non-spatial attribute values) efficiently. If certain changes occur, the trigger will be fired immediately and actions defined by the user will be taken. Moreover, this approach can be easily extended to handle reposted queries efficiently.

STING+ employs a set of sub-triggers to monitor the change of the data. It creates and removes these sub-triggers dynamically according to the insertions/deletions/updates in the system. Furthermore, STING+ evaluates these sub-triggers in a certain order to yield least cost. It is also shown via experiments that the STING+ has insignificant overhead.

References

[Agr95] Rakesh Agrawal and Giuseppe Psaila. Active data mining. *Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 3-8, 1995.

[Ank98] M. Ankerst, B. Braunmeller, H.-P. Kriegel, and T. Seidl. Improving adaptable similarity query processing by using approximations. *Proc. 24th Int. Conf. on Very Large Data Bases (VLDB)*, 1998.

[Est96] Martin Ester, Hans-Peter Kriegel, J. Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 226-231, 1996.

[Est97] Martin Ester, Hans-Peter Kriegel, and J. Sander. Spatial data mining: a database approach. *Proc. 5th Int. Symp. on Spatial Databases (SSD)*, 47-66, 1997.

[Est98a] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. *Proc. 24th Int. Conf. on Very Large Data Bases (VLDB)*, 1998.

[Est98b] M. Ester, A. Frommelt, H.-P. Kriegel, and J. Sander. Algorithms for characterization and trend detection in spatial databases. *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 1998.

[Fel97] Ronen Feldman, Yonatan Aumann, Amihood Amir, and Heikki Mannila. Efficient algorithms for discovering frequent sets in incremental databases. *Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, 59-66, 1997.

[Gal97] Antony Galton. Continuous change in spatial regions. *Proc. 3rd Int. Conf. on Spatial Information Theory (COSIT)*, 1-13, 1997.

[Gru98] Stephane Grumbach, Philippe Rigaux, and Luc Segoufin. The DEDALE system for complex spatial queries. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 213-224, 1998.

[Han97] Jiawei Han, Krzysztof Koperski, and Nebojsa Stefanovic. GeoMiner: a system prototype for spatial data mining. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 553-556, 1997.

[Hor97] K. Hornsby and M. J. Egenhofer. Qualitative representation of change. *Proc. 3rd Int. Conf. on Spatial Information Theory (COSIT)*, 15-33, 1997.

[Kno96a] Edwin M. Knorr and Raymond T. Ng. Finding aggregate proximity relationships and commonalities in spatial data mining. *IEEE Trans. on Knowledge and Data Engineering*, 8(6): 884-897, 1996.

[Kno96b] Edwin M. Knorr and Raymond T. Ng. Extraction of spatial proximity patterns by concept generalization. *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 347-350, 1996.

[Kno97] Edwin M. Knorr, Raymond T. Ng, and David L. Shilvock. Finding boundary shape matching relationships in spatial data. *Proc. 5th Int. Symp. on Spatial Databases (SSD)*, 29-46, 1997.

[Kop96] Krzysztof Koperski, Junas Adhikary, and Jiawei Han. Spatial data mining: progress and challenges. *SIGMOD Workshop on Research Issues on data Mining and Knowledge Discovery (DMKD)*, 1996.

[Ng94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. *Proc. 20th Int. Conf. on Very Large Data Bases (VLDB)*, 144-155,1994.

[Wan97] Wei Wang, Jiong Yang, and Richard Muntz. STING: a statistical information grid approach to spatial data mining. *Proc. 23rd Int. Conf. on Very Large Data Bases (VLDB)*, 186-195, 1997.
http://dml.cs.ucla.edu/∼weiwang/paper/VLDB97.ps

[Wan98] Wei Wang, Jiong Yang, and Richard Muntz. STING+: an approach to active spatial data mining. *UCLA CSD Technical Report #980031*, 1998.

[Wan98b] Wei Wang, Jiong Yang, and Richard Muntz. PK-tree: a spatial index structure for high dimensional point data. *Proc. 5th Int. Conf. on Foundation of Data Organizations (FODO)*, 1998.

[Wan99] Wei Wang, Jiong Yang, and Richard Muntz. STING+: an approach to active spatial data mining. *Proc. 15th Int. Conf. on Data Engineering (ICDE)*, 116-125, 1999.

[Wid96] Jennifer Widom and Stefano Ceri. *Active Database Systems*, Morgan Kaufmann, 1996.

[Zai98] Osmar R. Zaiane, Jiawei Han, Ze-Nian Li, Sonny H. Chee, and Jenny Chiang. MultiMediaMiner: a system prototype for multimedia data mining. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 581-583, 1998.

[Zan97] Carlo Zaniolo et al. *Advanced Database Systems*, Morgan Kaufmann, 1997.

[Zha96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 103-114, 1996.

A BNF of Trigger Language

```
<trigger>          ::= DEFINE [REPEAT] TRIGGER trigger-name
                       ON <on-clause>
                       [EVENTS <events>]
                       WHEN <when-clause>
                       [VALID time-interval]
                       DO <do-clause>
                       [SINCE time]
                       [UNTIL time]
<on-clause>        ::= <relations> | <classes>
<relations>        ::= relation-name | relation-name, <relations>
<classes>          ::= class-name | class-name, <classes>
<events>           ::= <event> | <event> AND <events>
<event>            ::= <simple-event> | <attr-trigger> | <region-trigger> |
                       <attr-δ-trigger> | <reg-δ-trigger> | . . .
<simple-event>     ::= INSERT | DELETE | UPDATE[(attribute)]
<do-clause>        ::= alarm | do-function | . . .
<when-clause>      ::= <attr-trigger> | <region-trigger> | <attr-δ-trigger> | <reg-δ-trigger> |
                       <when-clause> AND <when-clause>
<attr-trigger>     ::= SELECT <attr-clauses>
                       [WHERE <region-conds>]
                       [LOCATION <location>]
<region-trigger>   ::= SELECT REGION
                       [WHERE <region-conds>]
                       [LOCATION <location>]
<attr-δ-trigger>   ::= SELECT <stat-func>(attr-name)
                               {INCREASE | DECREASE | CHANGE} [<range> | <percent>]
                       [WHERE <region-conds>]
                       [LOCATION <location>]
<reg-δ-trigger>    ::= SELECT <reg-clause> [<range> | <percent>]
                       [WHERE <region-conds>]
                       [LOCATION <location>]
<attr-clauses>     ::= <attr-clause> | <attr-clause> AND <attr-clauses>
```

```
<attr-clause>    ::= <attr-func> [IN <range>]
<attr-func>      ::= attr-name [WITH <percent>] | <stat-func>(attr-name)
<stat-func>      ::= MAX | MIN | AVERAGE | SUM | ...
<range>          ::= RANGE <left-paren> number, number <right-paren>
<percent>        ::= PERCENT <left-paren> percentage, percentage <right-paren>
<reg-clause>     ::= {SIZE(REGION) | BOUNDARY(REGION)} CHANGE |
                     AREA(REGION) {INCREASE | DECREASE | CHANGE}
<location>       ::= <name-list> | <polygon-list> | (<region-query>)
<region-query>   ::= SELECT REGION
                     WHERE <region-conds>
                     [LOCATION <location>]
<region-conds>   ::= <region-cond> | <region-cond> AND <region-conds>
<region-cond>    ::= <density> | <attr-clauses> | <area> | <confidence>
<density>        ::= DENSITY IN <range>
<area>           ::= AREA IN <range>
<confidence>     ::= WITH CONFIDENCE percentage
<name-list>      ::= name | name; <name-list>
<polygon-list>   ::= <polygon> | <polygon>; <polygon-list>
<polygon>        ::= <points>
<points>         ::= <point> | <point>, <points>
<point>          ::= (coordinate, coordinate)
<left-paren>     ::= “[” | “(”
<right-paren>    ::= “]” | “)”
```