RC22947 (W0310-163) October 28, 2003
Computer Science

IBM Research Report

VCR Indexing for Fast Event Matching for Highly-Overlapping Range Predicates

Kun-Lung Wu, Shyh-Kwei Chen, Philip S. Yu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

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

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g, payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home.

VCR Indexing for Fast Event Matching for Highly-Overlapping Range Predicates

Kun-Lung Wu, Shyh-Kwei Chen and Philip S. Yu
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
{*klwu, skchen, psyu*}*@us.ibm.com*

Abstract

Fast matching of events against a large number of range predicates is important for many applications. An efficient predicate index is usually needed. It is difficult to build an effective index for multidimensional range predicates. It is even more challenging if the predicates are highly overlapping, as they usually are for many applications. We present a novel VCR indexing scheme for highly-overlapping 2D range predicates. VCR stands for *virtual construct rectangle*. Each predicate is decomposed into one or more VCRs, which are then considered to be activated. The predicate ID is then stored in the ID lists associated with these activated VCRs. Event matching is conceptually simple. For each event point, the search result is stored in the ID lists associated with the activated VCRs that cover the event point. However, it is computationally nontrivial to identify the activated covering VCRs. We define a *covering VCR set* for each event point and identify two important properties. Based on these properties, an efficient algorithm is presented for fast event matching. We also present an *overlapped decomposition* (OD) to reduce the storage overhead. Simulations are conducted to study the performance of VCR indexing.

Keywords: event matching, range predicate indexing, multidimensional access method, continual queries, pub/sub, virtual construct rectangles, VCR indexing.

1 Introduction

Fast matching of events against a large number of predicates is important for many applications, such as content-based publish/subscribe, continual queries, and moving object monitoring [1, 17, 7, 4, 18, 20]. Users simply specify their interests in the form of a conjunction of predicates. The system automatically monitors these user interests against a continual stream of events, or object locations.

Generally, an efficient predicate index is needed for fast event matching. Prior work for fast event matching has mostly focused on building predicate indexes with equality-only clauses [1, 12, 9]. However, many predicates contain non-equality range clauses. For example, user interests

(queries) are usually expressed as rectangles or circles in a 2D space in a moving object monitoring application [20]. With a range predicate index, we can quickly identify all the rectangles into which any object has just entered.

It is difficult to construct an effective index for multidimensional range predicates. It is even more challenging if these predicates are highly overlapping, as they usually are because people often share similar interests. For example, people tend to be interested in the current price ranges of individual stocks. As a result, the range predicates of their interests are likely to be highly overlapping. Although multidimensional range predicates can be treated as spatial objects, a typical spatial index, such as an R-tree or any of its variants, is generally not effective for fast event matching [14]. As has been pointed out in [14] and [10], this is because an R-tree quickly degenerates if spatial objects are highly overlapping [10, 11, 3, 22].

In this paper, we study the problem of fast event matching for highly-overlapping range predicates. We present and evaluate a novel VCR indexing scheme for 2D range predicates. VCR stands for *virtual construct rectangle*. A set of VCRs is defined for each data point within the attribute ranges. Each VCR has a unique ID and an associated predicate ID list. To insert a predicate, it is first decomposed into one or more VCRs, which become activated. The predicate ID is then inserted into each of the predicate ID lists associated with the decomposed and activated VCRs.

Event matching with VCR indexing is conceptually simple. An incoming event can be viewed as a point and event matching becomes finding all the activated VCRs covering that point. However, it is computationally nontrivial to identify all the VCRs that cover a point. In this paper, we define a *covering VCR set* for each point. These covering VCR sets share two important properties: *constant size* and *identical gap pattern*. Based on these two important properties, we present an efficient algorithm to identify all the activated VCRs that cover an event point.

We also present an *overlapped decomposition* (OD) method, as compared to *simple decomposition* (SD), to reduce the number of activated VCRs. OD uses VCRs of the same size to decompose a predicate, allowing overlapping among them. In contrast, SD uses VCRs of various sizes for decomposition. At first glance, this seems counterintuitive. One would think that substantial overlapping among decomposed VCRs is not a good idea. However, OD better facilitates the sharing of activated VCRs among predicates, hence reducing the total number of activated VCRs. As a result, we can use VCR hashing to effectively reduce storage requirement.

Simulations are conducted to study the performance of VCR indexing. The results show

that (1) the performance of VCR indexing depends on the maximal VCR size; (2) the concept of covering VCR set keeps the search time efficient; (3) with a moderate increase in storage, VCR indexing performs substantially better than an R-tree in search time, especially when the predicates are highly overlapping; and (4) overlapped decomposition makes VCR hashing more effective by reducing the number of activated VCRs.

The paper is organized as follows. Section 2 presents the details of VCR indexing. We first define a set of VCRs. Then, we describe various algorithms of VCR indexing, including decomposition, VCR hashing and event matching algorithms. We describe the concept of covering VCR set and the efficient computation of the VCRs in it. Section 3 shows the performance studies of VCR indexing. Section 4 discusses related work. Finally, Section 5 summarizes the paper.

2 VCR index for 2D range predicates

We examine the case where predicates are conjunctions of two intervals involving attributes X and Y. For simplicity, assume that the attribute ranges are $0 \leq X < R_x$ and $0 \leq Y < R_y$, respectively, and X and Y are integers. However, we only need the integer assumption to define a set of VCRs to be used for predicate decomposition. Event points can be non-integers. For example, in a 2D space, the spatial ranges for specifying user interests are defined with integers based on a virtual grid imposed upon a monitoring region, but object positions can be any non-integer. For applications where the ranges must be specified with non-integers, we can first expand the ranges to the nearest integers and then perform a check with the search result obtained from VCR indexing.

2.1 Virtual construct rectangles

We define a set of B VCRs for each point (x, y), where $0 \leq x < R_x$ and $0 \leq y < R_y$, and x and y are integers. The B virtual construct rectangles all have their bottom-left corners positioned at (x, y), but have different widths and heights. As a result, there are a total of BR_xR_y virtual construct rectangles.

Each of the B VCRs with the common bottom-left corner has its unique shape and size. Let (x, y, l_x, l_y) denote a VCR with the bottom-left corner positioned at (x, y) and top-right corner at $(x+l_x, y+l_y)$. Namely, the VCR has a width of l_x and a height of l_y. In this paper, we assume that $l_x \in \{1, 2^1, \cdots, 2^{k_x}\}$ and $l_y \in \{1, 2^1, \cdots, 2^{k_y}\}$, where $k_x = \log(L_x)$ and $k_y = \log(L_y)$ and L_x and L_y are the maximum width and height of a VCR, respectively. L_x and L_y are design parameters,

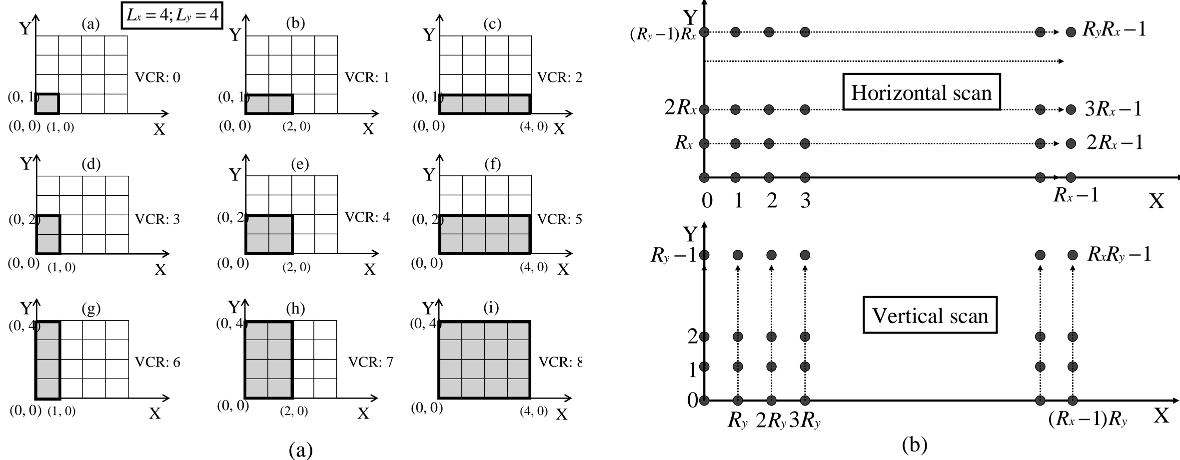


Figure 1: (a) An example of 9 VCRs with bottom-left corners at $(0,0)$; (b) Two approaches to scanning the bottom-left corners of VCRs.

which can be properly chosen based on system trade-offs. As a result, $B = (k_x+1)(k_y+1)$, or $(\log(L_x)+1)(\log(L_y)+1)$.

Fig. 1(a) shows an example of 9 VCRs with their bottom-left corners all positioned at $(0,0)$. In this example, $L_x = 4$ and $L_y = 4$. VCR 0 represents $(0,0,1,1)$ (see sub-plot (a) in Fig. 1(a)) while VCR 8 represents $(0,0,4,4)$ (see sub-plot (i) in Fig. 1(a)). Every one of the 9 VCRs is unique. Note that, in general, L_x and L_y need not be equal. These 9 VCRs will be repeated for each point. Each VCR has a unique ID and a predicate ID list associated with it.

Given a VCR $(x, y, 2^i, 2^j)$, where $0 \le i \le k_x$ and $0 \le j \le k_y$, one can compute a unique ID for it. Since there are B VCRs defined for each (x,y), where $0 \le x < R_x$ and $0 \le y < R_y$, the ID of any VCR depends on how we scan the bottom-left corners in the entire area bounded by the ranges of X and Y. Fig. 1(b) shows two examples of scanning: one is horizontal and the other is vertical. In this paper, we use the horizontal scanning shown in Fig. 1(b) to compute the unique ID for any given VCR. The case for vertical scanning can be similarly done. If the ID starts from 0, representing $(0,0,2^0,2^0)$ (see Fig. 1(a)), then the ID of any VCR $(x,y,2^i,2^j)$, is computed as follows:

$$\mathrm{ID}(x, y, 2^i, 2^j) = B(x + yR_x) + j(k_x + 1) + i, \tag{1}$$

where $B = (k_x+1)(k_y+1)$, $k_x = \log(L_x)$, and $k_y = \log(L_y)$.

Equation (1) can be easily proved. From the horizontal scanning in Fig. 1(b), there are $(x + yR_x)$ points before (x,y) in the horizontal scanning order. Hence, there are $B(x+yR_x)$ VCRs whose bottom-left corners are positioned on these $(x+yR_x)$ points. The second and third

Simple Decomposition:

```
SD((a, b, w, h)) {
    L_w = a; B_w = b; W_w = w; H_w = h; // workingRect = (a, b, w, h);
    decomposedSet = φ;
    while (H_w > 0) {
        H_s = maxVCRh(H_w);
        L_s = L_w; B_s = B_w; W_s = W_w; // stripRect = (L_s, B_s, W_s, H_s);
        while (W_s > 0) {
            find the largest VCR v such that
                (left(v) == L_s) ∧ (bottom(v) == B_s) ∧
                (width(v) ≤ W_s) ∧ (height(v) == H_s);
            decomposedSet = decomposedSet ∪ {v};
            L_s = L_s + width(v); W_s = W_s − width(v);
        }
        B_w = B_w + H_s; H_w = H_w − H_s;
    }
    return(decomposedSet);
}
```

Figure 2: Pseudo code for a simple decomposition of a predicate rectangle (a, b, w, h).

terms of Eq. (1) represent the ordering of the B VCRs that share the same bottom-left corner. It can be easily verified from sub-plot (a) in Fig. 1(a) to sub-plot (i) in Fig. 1(a).

2.2 Predicate insertion

To insert a user predicate p, defined as $a \leq p_x \leq a + w) \wedge (b \leq p_y \leq b + h)$, the corresponding predicate rectangle (a, b, w, h) is first decomposed into one or more VCRs. Note that (a, b, w, h) represents a rectangle whose bottom-left corner sits at (a, b), the width is w and the height is h. After decomposition, the predicate ID p is then inserted into each of the ID lists associated with those decomposed VCRs. In this section, we present two decomposition methods: simple decomposition and overlapped decomposition.

2.2.1 Simple decomposition

In simple decomposition, we first use the largest VCR that can fit into a predicate to decompose it. Then we use smaller VCRs to decompose the rest. The overlapping among decomposed VCRs is minimal, limiting only to the boundary lines among them.

Fig. 2 shows the pseudo code for simple decomposition (SD). SD first creates a working rectangle, which is initialized to be (a, b, w, h). Then, it cuts a strip rectangle from the bottom of the working rectangle and moves upwards until the working rectangle is completely removed. The width of a strip rectangle is w, the same as that of the input predicate, and the height is

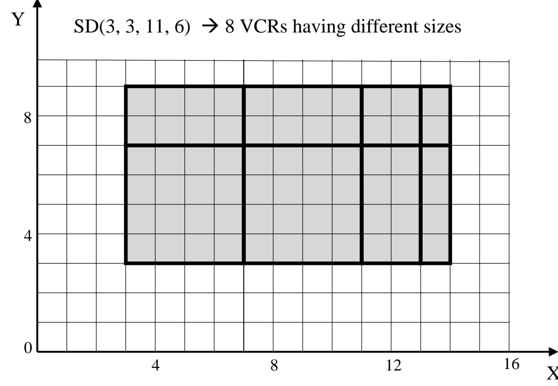


Figure 3: An example of simple decomposition.

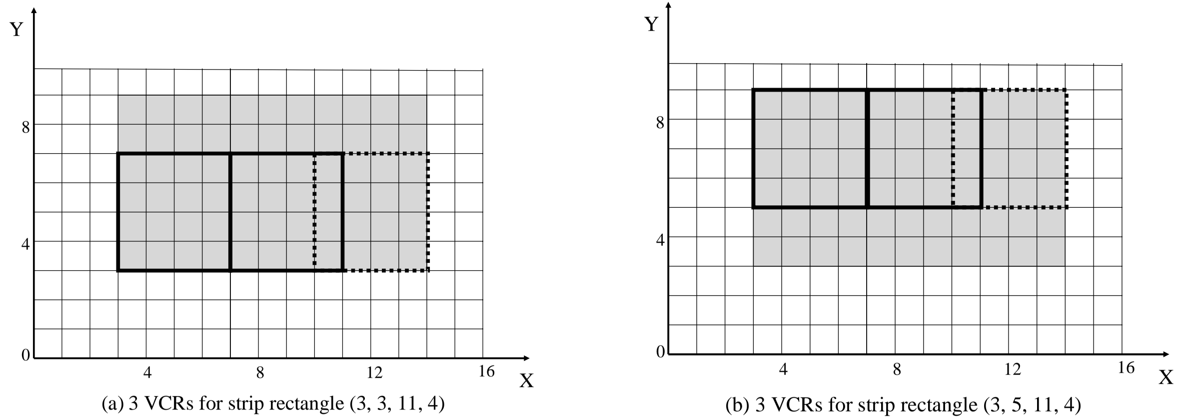
maxVCRh(H_w), the maximal VCR height that is less than or equal to H_w (the height of the working rectangle). As an example, if $H_w = 5$ and $L_y = 8$, then the height of a strip rectangle should be 4 because 4 is the maximal VCR height that is less than 5. On the other hand, if $H_w = 10$ and $L_y = 8$, then the height of a strip rectangle should be 8. For each strip rectangle, we find the largest VCR that has the same height as the strip rectangle and use that VCR to cut the strip rectangle. This process is repeated until the strip rectangle is completely removed. At the end, the set of decomposed VCRs is returned.

As an example, Fig. 3 shows the result of using SD to decompose a predicate rectangle $(3, 3, 11, 6)$. Assume that $L_x = 4$ and $L_y = 4$. First, it is partitioned into 2 strip rectangles: $(3, 3, 11, 4)$ and $(3, 7, 11, 2)$. Then each strip rectangle is decomposed into 4 VCRs each. Hence, $(3, 3, 11, 6)$ is decomposed into a total of 8 VCRs. These 8 VCRs have different sizes, 4×4, 4×2, 2×4, 2×2, 1×4 and 1×2. The overlapping among them is minimal. It occurs only on the boundary lines. For example, VCR $(3, 3, 4, 4)$ and VCR $(7, 3, 4, 4)$ overlap each other over 5 data points $(7, 3), (7, 4), (7, 5), (7, 6)$ and $(7, 7)$.

2.2.2 Overlapped decomposition

In SD, many small-sized VCRs may be activated by predicate decompositions (see Fig. 3). Hence, the average number of decomposed VCRs per predicate can be large. Since a predicate ID is inserted into each of the ID lists associated with the decomposed VCRs, storage requirement may become an issue. In order to reduce the number of decomposed VCRs, we also examine an overlapped decomposition.

In contrast to SD, the overlapped decomposition (OD) uses the same-sized VCR to decompose



(a) 3 VCRs for strip rectangle (3, 3, 11, 4)

(b) 3 VCRs for strip rectangle (3, 5, 11, 4)

Figure 4: An example of overlapped decomposition (a) and (b).

a given predicate. Basically, OD is very similar to SD in creating strip rectangles and using the largest VCR to decompose each strip rectangle. The difference between OD and SD is in how they handle the remnants of a strip rectangle and a working rectangle. More overlapping is allowed in OD. To achieve this, the left boundary of the last VCR is allowed to shift backward so that the same sized VCR is used in the decomposition. Similarly, the bottom of the last strip rectangle is allowed to shift downward so that the last strip rectangle has the same height as those of the other strip rectangles.

As an example, Figs. 4(a) and 4(b) show the result of using OD to decompose the same predicate rectangle $(3, 3, 11, 6)$. In contrast with Fig. 3, we only use a 4×4 VCR for the decomposition and there are only 6 decomposed VCRs (3 in Fig. 4(a) and 3 in Fig. 4(b)), instead of 8 as in SD. To clearly illustrate the overlapping among the decomposed VCRs, Fig. 4(a) shows that VCR $(7, 3, 4, 4)$ and VCR $(10, 3, 4, 4)$ overlap each other over 10 data points $(10, 3)$, $(10, 4)$, $(10, 5)$, $(10, 6)$, $(10, 7)$, $(11, 3)$, $(11, 4)$, $(11, 5)$, $(11, 6)$ and $(11, 7)$.

Note that both SD and OD decomposition methods allow overlapping among decomposed VCRs. The same predicate ID may appear in more than one ID lists in the search result. Hence, duplicate IDs may be reported.

Compared with SD, as will be shown in Section 3.5, OD better facilitates the sharing and reuse of decomposed VCRs among predicates. It reduces the number of activated VCRs. Less activated VCRs make it more effective to reduce the storage requirement via VCR hashing, which will be described next.

2.2.3 VCR hashing

A VCR is activated when it is used in the decomposition of a predicate which is inserted into the system. For each activated VCR, we maintain a predicate ID list. This list keeps all the IDs of predicates that use the VCR in their decompositions. In order to efficiently access this predicate ID list for a given VCR, we maintain an array of pointers, with the array index corresponding to the VCR ID. Predicate IDs are dynamically inserted into the corresponding predicate ID lists.

Because the total number of VCRs is BR_xR_y, the storage requirement for the array of pointers for maintaining the predicate ID lists can be large, especially if R_xR_y is large. Note that VCRs are virtual. We only need to maintain those ID lists associated with activated VCRs, not all of VCRs. In practice, the total number of activated VCRs is likely to be moderate. This is particularly true if predicates are highly overlapping. Overlapping predicates share common VCRs in their decompositions.

In order to reduce storage requirement, we use VCR hashing to maintain the predicate ID lists associated with activated VCRs. The ID of an activated VCR is hashed into a hash table of size H. In order to ensure even distribution of hash values, the VCR ID is first randomized and then divided by the hash size, a prime number, to obtain the hash location. The randomization step in the hash computation is important because activated VCRs tend to be clustered due to the fact that predicates are highly overlapping.

Once VCR hashing is employed, the search time inevitably slows down a bit because of the hash computation. But, it is a trade-off in order to limit the storage requirement. Moreover, efficient search time can be maintained by limiting hash collision to a minimal.

2.3 Matching an event against predicates

An event is equivalent to a data point. To find all the predicates matching an event is to find all the predicates whose IDs are stored in the ID lists associated with all the activated VCRs that cover that data point. In order to efficiently identify these activated VCRs for any given point, we first describe the concept of the covering VCR set for each point. More importantly, we then present an efficient algorithm to enumerate the IDs of all the VCRs in a covering VCR set.

2.3.1 Covering VCR set

Let $Cov(a, b)$ denote the covering VCR set of a point (a, b). $Cov(a, b)$ contains all the VCRs that can possibly cover (a, b). Graphically, $Cov(a, b)$ contains the set of VCRs that have bottom-left

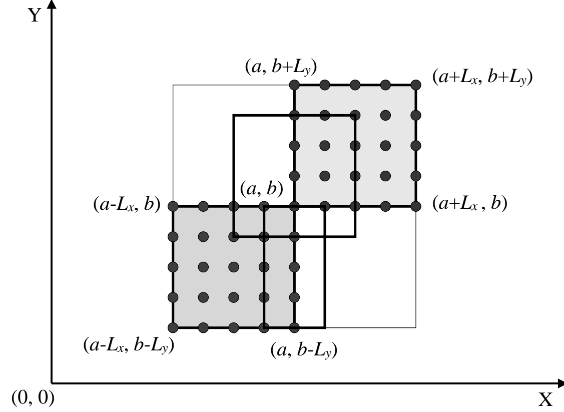


Figure 5: An example of the covering VCR set for data point (a, b).

corners in the shaded region southwest of (a, b), and upper-right corners in the shaded region northeast of (a, b) as shown in Fig. 5. Mathematically, $Cov(a, b)$ can be described as follows:

$$\begin{aligned} Cov(a,b) = \{(x, y, l_x, l_y) \quad | \quad & (a - L_x \leq x \leq a) \wedge (b - L_y \leq y \leq b) \wedge \\ & (a \leq x + l_x \leq a + L_x) \wedge (b \leq y + l_y \leq b + L_y)\}. \end{aligned} \tag{2}$$

The covering VCR sets share two important properties: *constant size* and *identical gap pattern.*

Constant size

For ease of discussion, we focus a search point (a, b) not under the boundary conditions, i.e., $L_x \leq a \leq R_x - L_x - 1$ and $L_y \leq b \leq R_y - L_y - 1$. The points under the boundary conditions, i.e., $0 \leq a < L_x$ or $R_x - L_x - 1 < a < R_x$ or $0 \leq b < L_y$ or $R_y - L_y - 1 < b < R_y$, can be similarly handled.

Theorem 1 *$|Cov(a, b)| = (2L_x + k_x)(2L_y + k_y)$, where $|Cov(a, b)|$ is the size of a covering VCR set for a data point (a, b) that is not under the boundary conditions.*

Intuitively, one may think that $|Cov(a, b)|$ is dependent on (a, b) and it is in the order of $L_x^2 L_y^2$, since, from Eq.(2), $Cov(a, b)$ depends on four variables, a, b, l_x and l_y. In fact, a naive algorithm to enumerate the IDs of all the VCRs in $Cov(a, b)$ might have the complexity in the order of $L_x^2 L_y^2$. However, a systematic counting reveals that $|Cov(a, b)| = (2L_x + k_x)(2L_y + k_y)$,

which is a constant and is independent of (a, b). Detailed proof of Theorem 1 can be found in Appendix A.

Identical gap pattern

Let $V_{i,(a,b)}$ denote the ID of a VCR covering (a, b) and $V_{i+1,(a,b)} > V_{i,(a,b)}$ for $1 \leq i < |Cov(a, b)|$. Namely, the VCR IDs are in a sorted order. The identical gap pattern property can be described as follows:

Theorem 2 *$V_{i+1,(a,b)} - V_{i,(a,b)} = V_{i+1,(c,d)} - V_{i,(c,d)}$, for $1 \leq i < |Cov(a, b)|$ and any two points (a, b) and (c, d).*

Theorem 2 can be verified by first grouping all the drawings in Fig 5 together as a unit and then moving it around. When the center is moved from (a, b) to another point (c, d), the relative positions of all the covering VCRs stay the same. Note that the gaps between different pairs of adjacent VCRs within a covering VCR set may not be the same. However, the gap pattern is identical across all covering VCR sets.

2.3.2 Search algorithm based on constant size and identical gap pattern

With Theorem 1, it is still computationally nontrivial to enumerate $Cov(a, b)$ during run time because of the many multiplications involved in computing a VCR ID (see Eq. (1)). Here, with the help of Theorem 2, we present an efficient algorithm that is based on simple additions, instead of complex multiplications.

For a data point (a, b), we first identify its unique *pivot VCR*, denoted as $PV(a, b)$ and its ID as $V_{PV(a,b)}$. This pivot VCR is defined as $(a - L_x, b - L_y, 1, 1)$. From Fig 5, $(a - L_x, b - L_y)$ is the bottom-left corner of VCR $V_{1,(a,b)}$, i.e., the covering VCR whose ID is the smallest. This pivot VCR has its bottom-left corner also positioned at $(a - L_x, b - L_y)$, but it has a unit width and a unit height. Note that the pivot VCR is not a member of $Cov(a, b)$.

$PV(a, b)$ can be used to compute a distance table,

$$DT(a, b) = \{V_{i,(a,b)} - V_{PV(a,b)} | 1 \leq i \leq |Cov(a, b)|\}.$$

$DT(a, b)$ contains the ID differences between all the covering VCRs and $PV(a, b)$. From Theorem 2, we can show that $DT(a, b) = DT(c, d)$ for any (a, b) and (c, d). Namely, the relative distances between the pivot VCR and the VCRs within a covering VCR set remain the same for all points.

As a result, we can pre-compute a single distance table DT and store it in main memory. The storage cost for storing this DT is minimal.

With the use of PV and a pre-computed DT, the event matching process is truly simple and efficient. Given an event point (a, b), we first compute the ID of $PV(a, b)$, denoted as c. Then we sequentially scan the distance table DT. For each value d encountered in DT, the ID $c + d$ corresponds to a unique covering VCR for (a, b). Therefore, we effectively enumerate $Cov(a, b)$ in a single pass with simple additions during the search time.

2.4 Predicate deletion

To delete a predicate from the system, it is decomposed into one or more VCRs, similar to predicate insertion. Then, the predicate ID is removed from each of the ID lists associated with the decomposed VCRs. In this paper, we assume that a single linked ID list is used and each new ID is simply inserted at the head of a predicate ID list. This approach makes predicate insertion very efficient. However, it may slow down predicate deletion. On average, it needs to traverse half of the list. Other data structure certainly can be used to maintain the ID lists if deletion time is important. However, this is not the focus of this paper.

3 Performance analysis

3.1 Simulation studies

Simulations were conducted to evaluate and compare VCR indexing with an R-tree. We implemented various VCR indexing schemes as described in Section 2. For the case of an R-tree, we downloaded an implementation by Y. Theodoridis [23] and used it to read and process our input data.

Two different values for $R_x R_y$ were considered: 60,000 (moderate) and 3,000,000 (large). The degree of predicate overlapping was modeled as a skewed rule, $\alpha - \beta$. Namely, α portion of the predicates have their bottom-left corners uniformly chosen from a rectangle whose area is β portion of $R_x R_y$. For simplicity, the center of this rectangle approximately coincides with that of the range rectangle $\{(x, y)|0 \leq x < R_x; 0 \leq y < R_y\}$. Once the bottom-left corner was chosen, the width w of a predicate was chosen uniformly from $[1, W_x]$ and the height h was chosen uniformly from $[1, W_y]$. We varied W_x, W_y, L_x, and L_y and the overall messages remained the same. For most of the experiments, $W_x = 40, W_y = 20, L_x = 16$, and $L_y = 8$.

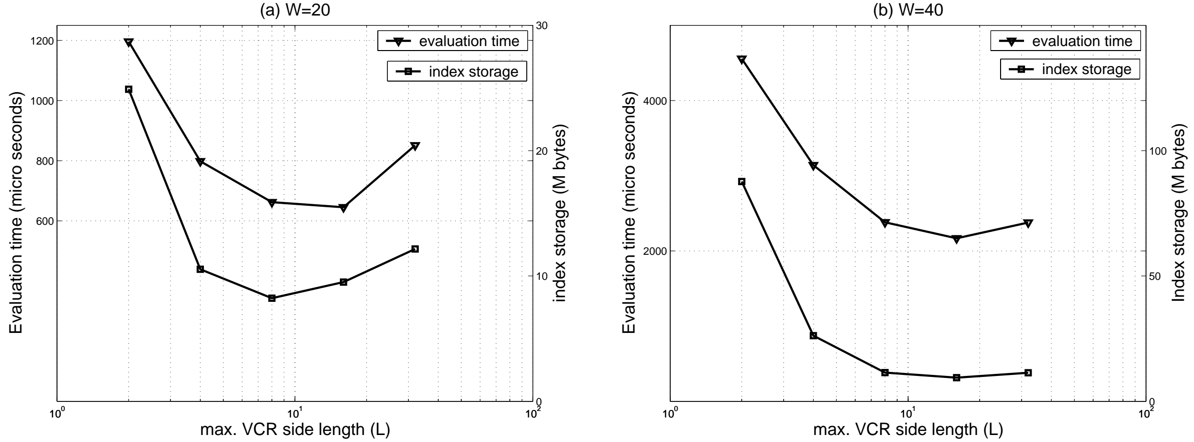


Figure 6: The impacts of maximal VCR side length when (a) W=20; (b) W=40.

A total of n predicates were generated and inserted into the VCR index. We varied n from 32,000 to 256,000. For VCR indexing, each predicate was decomposed into one or more VCRs based on the decomposition algorithm, and the predicate ID was inserted into each of the predicate ID lists associated with the decomposed VCRs. For the case of a moderate R_xR_y (60,000), we maintained a header pointer for each VCR, regardless of its activation status. For the case of a large R_xR_y (3,000,000), we use a hash table to maintain only the activated VCRs.

After inserting n predicates, we performed 100,000 searches, with the event points chosen based on the same $\alpha - \beta$ rule. The average search time was obtained from the simulation. The storage requirement is computed based on the memory needed to maintain the array of pointers if no VCR hashing is used (or the hash table) and the actual predicate ID lists. The actual storage requirement for the predicate ID lists was obtained from the simulation.

3.2 The impact of VCR size

The maximal VCR size has important performance impacts, both on the average search time and the total storage cost. Let the maximal size length of a VCR be L. Namely, $L_x = L_y = L$. If L is too small, a large number of small VCRs will be activated, increasing search time and storage cost. If L is too large, both the total number of VCRs and the size of a covering VCR set will be large, also increasing the search time and storage cost. The optimal L depends on the workload, especially the distribution of predicate sizes.

Fig. 6(a) and Fig. 6(b) show the impacts of the maximal VCR side length L on the average search time and storage cost when $W_x = W_y = W = 20$ and 40, respectively. For this experiment,

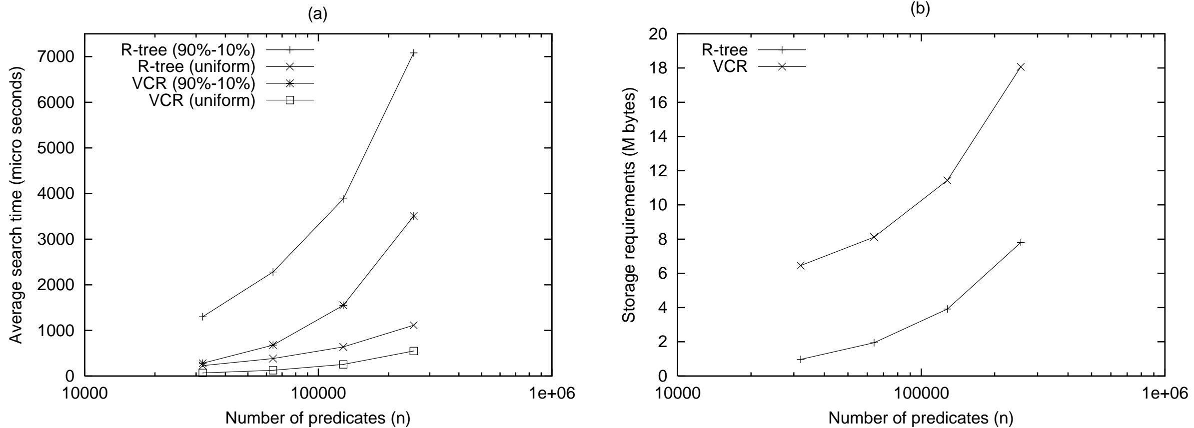


Figure 7: Comparison of an R-tree and VCR in terms of (a) average search time; (b) storage requirement.

a moderate R_xR_y was used. For both figures, the left y-axis was used for average search time of an event matching and the right y-axis was used for the index storage requirement. For the case of $W = 20$, the optimal L is 8 in terms of storage cost. However, the optimal L is 16 in terms of average search time. For the case of $W = 40$, the optimal L is 16 in terms of storage and search time. In general, the larger the average size of predicate ranges is, the larger the optimal VCR side length becomes. For the experiments in the rest of the paper, we chose $L = 8$ for the case of $W = 20$ and $L = 16$ for the case of $W = 40$.

3.3 Comparison of VCR with R-tree

In this section, we compare VCR with an R-tree under a moderate R_xR_y. Fig. 7(a) shows the average search times of VCR indexing and R-tree indexing with different degrees of predicate overlapping. Predicates overlap more for the 90%-10% case and they overlap less for the uniform case. Simple decomposition was used for the VCR indexing schemes.

Fig. 7(a) shows that the search time of VCR is substantially better than R-tree, especially when the predicates are highly overlapping. For instance, even for a small n (32,000), the average search times are $1,300\mu$ and 304μ seconds for the 90%-10% and uniform cases, respectively. In comparison, the search times of VCR indexing are 280μ and 60μ seconds, respectively. For a large n, the performance difference between VCR and an R-tree is more substantial. Fig. 7(a) clearly shows that an R-tree is not effective for handling range predicates that are highly overlapping. Note that, with highly overlapped predicates, the minimum bounding rectangles in the internal nodes of an R-tree are highly overlapped. Thus, the search in an R-tree quickly degenerates into

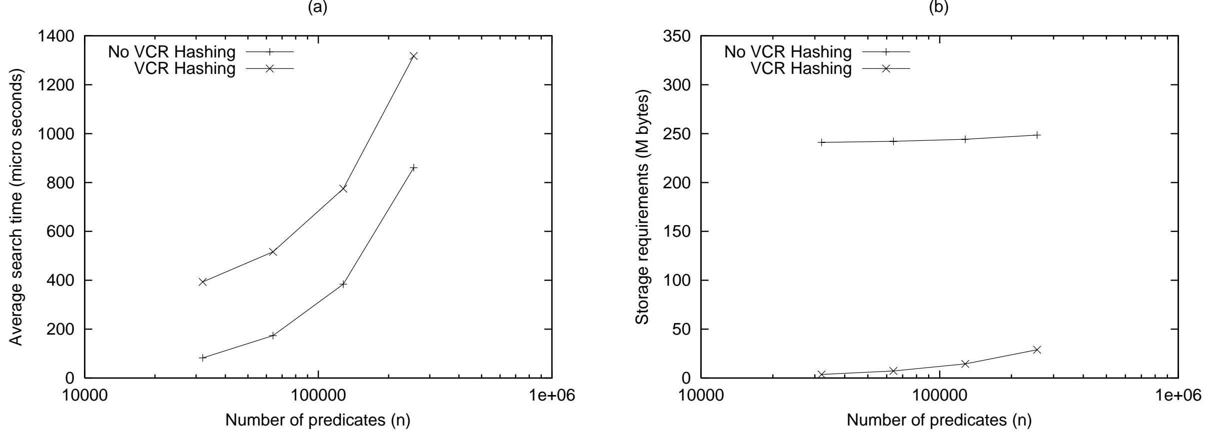


Figure 8: The impact of VCR hashing on (a) average search time; (b) storage requirement.

almost a full-tree traversal.

The efficient search time of VCR indexing in Fig. 7(a) is achieved with an increase in storage cost. However, such an increase is rather modest. Fig. 7(b) shows the corresponding storage requirements of an R-tree and VCR. Note that the storage requirement does not depend on the degree of predicate overlapping. Hence, we only show one line for each indexing scheme.

3.4 The impact of VCR hashing

As $R_x R_y$ becomes large, the storage requirement for the static array of pointers that maintain the predicate ID lists can become quite large. When predicates are highly overlapping, there is only a small number of activated VCRs. To reduce the storage, we eliminated the static array and use a smaller hash table to maintain only the activated VCRs. Fig. 8(b) shows the storage requirement of two VCR indexing schemes, one with and the other without VCR hashing. For this experiment, $R_x R_y = 3,000,000$. The hash table size was set to be about $10n$. Overlapped decomposition was used. Fig. 8(b) demonstrates that VCR hashing is effective in reducing the storage requirement.

The reduction in storage cost in Fig. 8(b) is achieved with a trade-off in search time in Fig. 8(a). This is because we need to do a hash computation for each VCR in the covering VCR set. For this experiment, the search times for VCR with hashing are between 400μ and $1,300\mu$ seconds with highly overlapped predicates (90%-1%), as n varies from 32,000 to 256,000. The search times without VCR hashing are between 80μ and 860μ seconds, respectively. Nevertheless, we think this is a good trade-off, especially for a relatively large $R_x R_y$ and relatively small n.

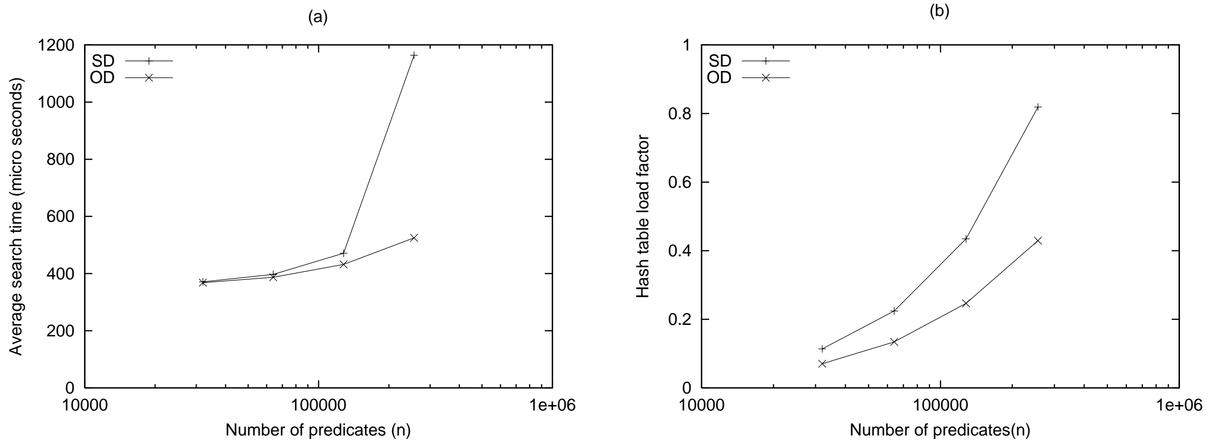


Figure 9: The impact of overlapped decomposition on (a) average search time; (b) hash table load factor.

3.5 The impact of overlapped decomposition

Compared with simple decomposition, overlapped decomposition algorithm activates less number of VCRs. Figs. 9(a) and 9(b) compare OD and SD in average response time and hash table load factor, respectively. Predicates were highly overlapping following a 90%-10% skew rule. For this experiment, we fixed the hash size at 1,800,017. As n increases, the number of unique activated VCRs increases, increasing the probability of hash collision. The average response time is much better using OD than using SD, especially when n is large (see Fig. 9(a)). This is due to the fact that SD tends to activate more smaller sized VCRs than OD. As n increases, these extra activated VCRs in SD make the hash table load factor go higher, causing more hash collision.

4 Related work

There are strong interests in event matching schemes for content-based pub/sub systems [1, 19, 9, 5] and triggers [12]. There are roughly two kinds of pub/sub algorithms. The first kind of schemes consists of a single phase. For example, the Gryphon system builds a search tree with subscription predicate clauses [1]. However, no non-equality predicate clauses were considered in [1]. A second kind of event matching algorithms involves two phases [19, 9]. The predicate clauses are tested in the first step, and then the matching subscriptions are computed using the results from the first step.

Recently, there has been research work on selective dissemination system that can handle subscriptions written in XPath for XML documents [2, 8, 6, 16]. XPath queries were converted

into various data structures which react to XML parsing events. In contrast, the event matching problem discussed in this paper is for simpler subscriptions that contain conjunction of predicate intervals.

Continual queries [7, 17, 12] have been developed to permit users to be notified about changes that occur in the database. They evaluate conditions that combine incoming event with predicates on a current database state. This makes it difficult for these systems to scale over hundreds of thousands of queries because they must check hundreds of thousands of complex conditions each time a new event modifies the database state. In contrast, we focus on fast event matching for highly overlapping multidimensional range predicates.

There are various spatial data structures that can handle non-equality predicates, such as R-trees and its variants [21, 10]. With R-trees, subscriptions would be represented as regions in a multidimensional space. As pointed out in [14, 10], R-trees are generally not suitable for event matching because R-trees quickly degenerate if the objects are highly overlapping. Many data structures have been designed to index a list of predicate intervals defined on the same attribute, such as segment trees [21], interval binary search trees [13], and interval skip lists [14]. But, these are mainly for the case of single dimension.

Note that the VCRs defined in this paper are different from the space-filling curves, such as the Hilbert curve and the Z-ordering [10], that are used to store multidimensional point data. In a space-filling curve, the universe is first partitioned into grid cells. Each of the grid cells is labeled with a unique number that defines its position in the total order of the space-filling curve. The objective is to preserve spatial proximity in the original point data. In contrast, a set of VCRs is defined for each point. These VCRs are used to decompose predicates, which are spatial objects such as rectangles. Furthermore, VCRs are not designed to partition the universe.

The SR-tree presented in [15] is the most related to our VCR indexing in that both are designed for multidimensional interval data. However, they are designed to handle different specific workload issues. SR-tree is to tackle skew in interval sizes while VCR indexing to tackle predicate overlapping. SR-tree is a modified R-tee. The emphasis of the SR-tree is to improve the performance of an R-tree under the workload where the interval sizes are skewed. There are large-sized intervals among otherwise small-sized intervals. By mixing large-sized intervals with small-sized ones, the minimum bounding rectangles used in an R-tree are unduly enlarged by the large-sized intervals. Enlarged bounding rectangles quickly degrade the performance of an R-tree index. The SR-tree moves the large-sized intervals from the leaf-nodes into the internal

nodes of an R-tree. However, the SR-tree still does not solve the issue of interval overlapping among the small-sized intervals.

5 Summary

In this paper, we have presented and evaluated a novel VCR indexing scheme for fast event matching for highly-overlapping 2D range predicates. It uses a novel concept called virtual construct rectangle. A set of VCRs is defined for each data point within the attribute ranges. Each VCR has a unique ID and a predicate ID list. A predicate is first decomposed into one or more VCRs. Then, the predicate ID is inserted into each of the predicate ID lists associated with the decomposed VCRs. A VCR is activated if a predicate using the VCR in its decomposition is inserted into the system. Event matching is conceptually simple. For each event point, the search result is already stored in the ID lists associated with activated VCRs covering that point. However, it is nontrivial to identify all the covering VCRs for any point.

We have defined a covering VCR set for each point and identified two important properties among all the covering VCR sets: constant size and identical gap pattern. Based on these two properties, we have described an efficient method that first identifies a pivot VCR for a point. The ID of this pivot VCR is then added to a pre-computed distance table which contains the ID differences between all the covering VCRs and the pivot VCR.

We have presented an overlapped decomposition method to reduce the number of activated VCRs. As attribute ranges become large, the storage requirement for a straightforward implementation of VCR indexing can become quite large. Overlapped decomposition facilitates the sharing and reuse of activated VCRs. With less activated VCRs, VCR hashing becomes more effective in reducing the storage requirement.

Simulations have been conducted to study the performance of VCR indexing. The results show that (1) the performance of VCR indexing is dependent on the maximal VCR side length; (2) the concept of covering VCR set and a pre-computed distance table make the search time of VCR indexing very efficient; (3) with a modest increase in storage, VCR hashing performs significantly better than an R-tree in search time, especially when the predicates are highly overlapping; and (4) overlapped decomposition makes VCR hashing more effective, especially when the attribute ranges are large.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Proc. of Symp. on Principles of Distributed Computing*, 1999.

[2] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proc. of VLDB*, pages 53–64, 2000.

[3] N. Beckmann and H.-P. Keigel. The R^*-tree: An efficient and robust access method for points and rectangles. In *Proc. of 1990 ACM SIGMOD*, pages 322–331, 1990.

[4] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams – a new class of data management applications. In *Proc. of VLDB*, 2002.

[5] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving expressiveness and scalability in an Internet-scale event notification service. In *Proc. of Symp. on Priciples of Distributed Computing*, pages 219–227, 2000.

[6] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *Proc. of ICDE*, pages 235–244, 2002.

[7] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for Internet databases. In *Proc. of ACM SIGMOD*, pages 379–390, 2000.

[8] C.-W. Chung, J.-K. Min, and K. Shim. APEX: An adaptive path index for XML data. In *Proc. of ACM SIGMOD*, 2002.

[9] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proc. of the ACM SIGMOD*, 2001.

[10] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[11] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of ACM SIGMOD*, 1984.

[12] E. Hanson, C. Carnes, L. Huang, M. Konyala, L. Noronha, S. Parthasarathy, J. B. Park, and A. Vernon. Scalable trigger processing. In *Proc. of ICDE*, pages 266–275, 1999.

[13] E. Hanson, M. Chaaboun, C.-H. Kim, and Y.-W. Wang. A predicate matching algorithm for database rule systems. In *Proc. of ACM SIGMOD*, pages 271–280, 1990.

[14] E. Hanson and T. Johnson. Selection predicate indexing for active databases using interval skip lists. *Information Systems*, 21(3):269–298, 1996.

[15] C. P. Kolovson and M. Stonebraker. Segment indexes: Dynamic indexing techniques for multi-dimensional interval data. In *Proc. of ACM SIGMOD*, 1991.

[16] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *Proc. of VLDB*, 2001.

[17] L. Liu, C. Pu, and W. Tang. Continual queries for Internet scale event-driven information delivery. *IEEE TKDE*, 11(4):610–628, July/Aug. 1999.

[18] S. R. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proc. of ACM SIGMOD*, 2002.

[19] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for Web-based publish/subscribe systems. In *Proc. of Int. Conf. on Cooperative Information Systems*, pages 162–173, 2000.

[20] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query indexing and velocity constrainted indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. on Computers*, 51:1124–1140, Oct. 2002.

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

[22] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. In *Proc. of 13th VLDB Conference*, pages 507–518, 1987.

[23] Y. Theodoridis. C implementations of the R-tree, R+-tree, and R*-tree. http://www.dbnet.ece.ntua.gr/~theodor/files/rtrees/.

Appendix A

Theorem 1 *$|Cov(a,b)| = (2L_x + k_x)(2L_y + k_y)$, where $|Cov(a,b)|$ is the size of a covering VCR set for a non-boundary data point (a,b).*

Consider the shaded region southwest of (a,b) in Fig. 5. For any given y within the region, there are $L_x + 1$ points, each can be the bottom-left corner of B VCRs. Under the same height h and along the same y, each of the $L_x + 1$ points has $(k_x + 1)$ VCRs that can possibly cover (a,b). Due to the distances to (a,b), however, there are $L_x/2^i$ points contributing i VCRs each to $Cov(a,b)$. The widths of these VCRs are L_x, $L_x/2$, $\cdots$, $L_x/2^{i-1}$, where $i = 1, 2, \cdots, k_x$. There are 2 remaining points, $(a-1, y)$ and (a, y), each of which can have $k_x + 1$ covering VCRs (of width $1, 2^1, \cdots, 2^{k_x}$). Summing them up, we have the term $2L_x + k_x$. Similarly, by varying the height h, we can calculate along the y dimension and obtain the term $2L_y + k_y$. $|Cov(a,b)|$ is the product of both terms.

For example, as shown in Fig. 5, under the same height ($L_y = 4$) and the same y ($b-4$), the 2 points $(a-4, b-4)$ and $(a-3, b-4)$ contribute 1 VCR (of width 4) each to $Cov(a,b)$. Point $(a-2, b-4)$ contributes 2 VCRs (of widths 4 and 2) to $Cov(a,b)$. Finally, the 2 points $(a-1, b-4)$ and $(a, b-4)$ contribute 3 VCRs (of widths 1,2, and 4) each to $Cov(a,b)$. Thus, a total of 10 VCRs with the same height and y cover point (a,b).