Optimal Partitioning for Efficient I/O in Spatial Databases*

Hakan Ferhatosmanoglu, Divyakant Agrawal, and Amr El Abbadi

Computer Science Department, University of California at Santa Barbara
{hakan,agrawal,amr}@cs.ucsb.edu

Abstract. It is desirable to design partitioning techniques that minimize the I/O time incurred during query execution in spatial databases. In this paper, we explore optimal partitioning techniques for spatial data for different types of queries, and develop multi-disk allocation techniques that maximize the degree of I/O parallelism obtained during the retrieval. We show that hexagonal partitioning has optimal I/O cost for circular queries compared to all possible non-overlapping partitioning techniques that use convex regions. For rectangular queries, we show that although for the special case when queries are rectilinear, rectangular grid partitioning gives superior performance, hexagonal partitioning has overall better I/O cost for a general class of range queries. We then discuss parallel storage and retrieval techniques for hexagonal partitioning using current techniques for rectangular grid partitioning.

1 Introduction

Spatial databases and Geographical Information Systems(GIS) have gained in importance by the recent developments in information technology. In these applications, the data objects are represented as two-dimensional feature vectors, and the similarity between objects are defined by a distance function between corresponding feature vectors. Several index structures have been proposed for retrieval of spatial data [4,8]. Most of these approaches are based on data partitioning with a rectangular organization. Grid based file structures have been effectively used to index spatial data, and there have been several approaches based on the grid partitioning. Because of their simplicity in hashing and mapping to physical storage, *regular* equi-sized partitioning are widely used for retrieval and storage of spatial data. A common example of such techniques is the regular grid partitioning. One very important application of regular grid partitioning is multi-disk *declustering* of spatial data. First, the data space is partitioned into disjoint regular rectangular partitions. Then the data partitions or buckets are allocated to multiple I/O devices such that neighboring partitions are allocated to different disks. Performance improvements for queries occur when the buckets involved in query processing are stored on different disks, and hence can

* This work was partially supported by NSF grant EIA98-18320, IIS98-17432 and IIS99-70700.

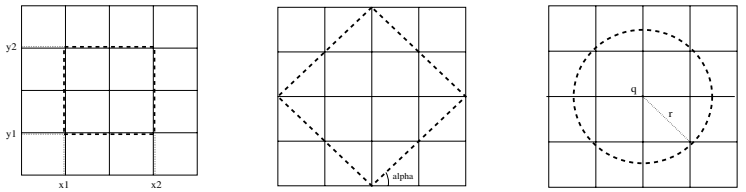
R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 889–900, 2001.

be retrieved in parallel. Numerous declustering methods using *non-overlapping* rectangular partitioning have been proposed [1,7]. Another successful application of rectangular partitioning is the vector approximation (VA) based indexing for multi-dimensional data [9]. The VA-file approach divides the data space into rectangular partitions, and each data point is represented by the bit representation of the corresponding partition. In several similar applications, it is desirable to design partitioning techniques that tile the data space without holes and overlaps. These techniques have simple hashing schemes, and they don't have the problems caused by high number of overlaps between partitions which result severe degradation in the query performance.

Minimizing the number of I/O operations in query processing is crucial for fast response times. This cost can be reduced by reducing the expected number of page retrievals during the execution of queries. In a range query, the pages that may have the query result need to be retrieved from secondary storage. As we will establish, the expected number of page retrievals directly depends on the underlying technique that is used to organize the data set, i.e., partitioning of the data [2]. It is therefore very important to develop partitioning techniques which minimize the expected number of partitions retrieved by a query and hence the number of disk accesses. It is well known that rectangles are effective for non-overlapping partitioning. An important question to ask is whether rectangular grid based partitioning is the optimal approach for non-overlapping partitioning to minimize the I/O cost?

In this paper, we explore optimal partitioning techniques for different types of queries. In particular, we show that hexagonal partitioning has optimal I/O cost for circular queries compared to all possible non-overlapping partitioning techniques that use convex regions. For rectangular queries, we show that although for the special case when queries are rectilinear rectangular grid partitioning gives superior performance, hexagonal partitioning has overall better I/O cost for a general class of range queries. We also develop simple techniques for the storage and retrieval of hexagonal partitioning by applying the techniques developed for rectangular grid partitioning. The techniques developed in this paper considers the objectives that are crucial for multi-disk searching: i) minimizing the number of page accesses during the execution of the query, ii) maximizing the I/O parallelism, iii) minimizing the disk-arm movement (seek time).

In Section 2 we discuss the importance of data organization on minimizing the I/O cost of spatial queries and summarize alternative partitioning techniques. In Section 3, we analyze the partitioning techniques with respect to the I/O cost of spatial queries and show the optimal partitioning technique for each query type. Section 4 discusses techniques for storage and retrieval of hexagonal partitioning using current techniques for rectangular grid. Section 5 includes the final discussion and conclusions.



(a) Rectilinear (b) Diamond (c) Circular

Fig. 1. Range queries intersecting all partitions

2 Data Organization for Efficient Spatial Queries

The two most common spatial queries are range queries and similarity queries. In a range query, the user specifies an area of interest and all data points in this area are retrieved. In a similarity query, the query point is specified and all points "similar" to the query point are retrieved. A popular related query is k-nearest neighbor query. The k-nearest neighbor, $k-NN$, problem is defined as finding the k nearest points to the query point q. Traditional range queries have been specified using rectilinear queries, i.e., rectangular with sides parallel to axes. A rectilinear range query $Q_r = ([a_1, b_1], [a_2, b_2])$ specifies a range of values for each dimension. The result of the query is the set of all data objects that have values within the specified range in each dimension. More generally, a rectangular query can be defined as a rectilinear rectangle with a rotation angle of α with respect to the $x-$axis. For example, a rectilinear square range query with a rotation of $\pi/4$ gives us a diamond query (see Figure 1(b)). A commonly used query is the circular range query which is also called as ϵ-similarity query. It specifies a query point q and and a radius r, $Q_c = (q, r)$, which defines the acceptable region of similarity. All data objects that fall into the circle defined by the pair (q, r) are in the answer set. It is interesting to note that range queries of different shapes correspond to different types of similarity queries using different metrics. For example, the circular range query corresponds to similarity in L_2 metric, the rectilinear rectangular range query corresponds to similarity in the L_∞ metric and the diamond range query corresponds to similarity in the L_1 metric.

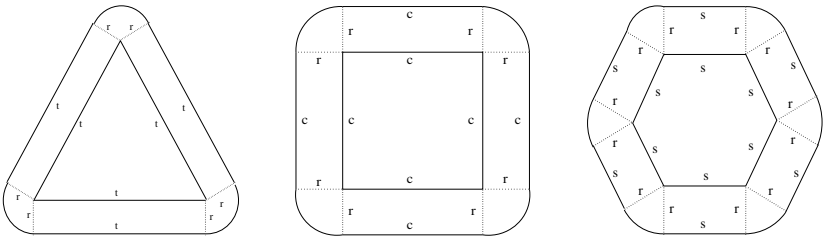
In both range and similarity queries, the pages that have the possibility to contain a portion of the query result are retrieved as a result of the query. As mentioned before, the spatial data objects are grouped according to their spatial locations and they are stored as pages in physical storage. Each page represents a spatial location in the data space. Therefore, in range queries the pages that are intersected by a query are retrieved as a result of the query. Similarly, $k-NN$ queries have to retrieve all pages that intersect the circle with the query point as the center and the distance between the query point and the $k-NN$ point as the radius. Even if there is a small portion of intersection of a page with the query region, the page needs to be retrieved since it may contain relevant data.

Is it possible to reduce this I/O cost by changing the initial page organization of the data space? What is the best possible page organization? $\lceil \frac{QueryResultSize}{PageCapacity} \rceil$ is the minimum number of page accesses that is needed to answer any query on any data sets. Although, it is not possible to develop a technique that achieves this minimum cost for every possible range query, a careful investigation is needed to reduce the I/O cost as much as possible. In general, assuming a uniformly distributed data space of area 1, the optimal cost for a range query with area A is equal to $\lceil A.p \rceil$, where p is the total number of pages. The number of pages accessed plays an important role in the response time incurred by range queries. The key point to minimize the I/O cost for queries depends on how data points are organized into pages. If a query is executed in two differently organized copies of the same data set, different numbers of pages will be retrieved depending on the way the data-space is initially organized. For efficient queries, the underlying organization of the data must be designed to reduce the expected number of pages retrieved by the queries. An organization of a database is *I/O optimal* if and only if it minimizes the expected number of page accesses incurred by the queries.

Regular equi-sized partitioning are widely used for organization of spatial data. They provide very efficient functions for hashing and mapping the partitions to pages in storage. The general approach is first to tile the data space into disjoint regular rectangular partitions and then to map each partition to a physical page in the storage. Are there any alternative techniques that tile the data space with non-overlapping partitions of identical shapes? It is known that it is not possible to tile the space with regular non-overlapping identical convex polygons with edges of more than 6 and there are only three basic tile shapes (triangles, rectangles, and hexagons) for regular partitioning [3]. We will consider these three basic shapes and compare their behavior under various conditions. Equilateral triangle, square, and regular hexagon are considered for triangle, rectangle, and hexagon respectively. They all have simple hashing properties that can be used to map the partitions to physical storage easily. We note that, although we consider these possible partitionings for our analysis, the optimality results in this paper are not restricted to these shapes.

3 I/O Cost for Various Query Types

Given a query, we explore different partitioning techniques that minimize the expected number of pages retrieved. The partitions that have to be accessed are the ones that intersect the range query region. Therefore, we will compute the expected number of partitions that intersect a range query region by using the methodology of Minkowski sum, $MSum$ [9]. In general, the Minkowski sum of two closed regions can be considered as expanding one region by the other. To compute the expected number of partitions that intersect a query, first the region of a partition, which also corresponds to a page region, is enlarged by considering the shape of the query. Suppose the partitions, which also correspond to the pages, are square shapes of side length c and the queries are circular.



(a) Triangle-Circle (b) Square-Circle (c) Hexagon-Circle

Fig. 2. Minkowski Sums w.r.t. Circles

The Minkowski sum methodology is used to compute the expected number of partitions that intersect the circular query region as follows. The Minkowski sum of a square page of side length c, with respect to a circular region with a radius of r, is the enlarged object with area $MSum(square_c, circle_r)$. The enlarged object is created by moving the center of the circular query over the surface of the square (Figure 2(b)). Therefore, it consists of all points that are either in the square page or have a distance to the surface of the page less than r. The points within this enlarged object correspond to the center of all possible circular queries that intersect this page. Assuming uniform distribution, the fraction of the area of the enlarged object to the area of the data space is the probability that the corresponding page is being accessed, $P(page)$. Considering the unit data space, i.e., $[0, 1]^2$, or simply normalizing the areas with respect to the total area, the area of the enlarged object gives the probability of the page to be accessed. Note that, the regions that are created by the Minkowski sum of the partitions in the border of the two-dimensional data space are negligible when compared to the total amount of regions created by Minkowski sums of all partitions. The expected number of pages which are intersected by a query q is $E(q)$ and it is the sum of the probabilities of the pages intersected by the query. If we have regular equi-sized partitioning, i.e., p partitions of identical shape, then the expected number of intersected partitions by the query is p times the probability of a page being accessed, i.e., $E(q) = p \cdot P(page)$. Therefore, for a partitioning scheme that tiles the space with pages of shape s, the expected numbers of intersected partitions for a query q of any shape is $E_s(q) = p \cdot MSum(s, q)$, where p is the number of partitions. For example, for a square partitioning this equation is simply $E_{square}(q) = p \cdot MSum(square_c, q)$. In this section, we analyze the I/O cost performance of the partitioning techniques for range queries. We will investigate the optimal partitioning, assuming uniform distribution, that must be used for different conditions to minimize the I/O cost.

3.1 I/O Cost for Circular Range Queries

To analyze the performance for circular range queries, we first compute the Minkowski sums of triangle, square, and hexagon shapes with respect to a circular region of radius r. Figure 2 shows the Minkowski sum regions for triangular, square, and hexagonal pages, i.e., $MSum(triangle_t, circle_r)$, $MSum(square_c, circle_r)$, and $MSum(hexagon_s, circle_r)$, respectively. The values of each edge, t, c, and s, are assigned such that the area of each triangle, square, and hexagon partition is equal to $1/p$, i.e., $\frac{t^2\sqrt{3}}{4} = c^2 = s^2\frac{3\sqrt{3}}{2} = 1/p$. As can be seen from the figures, the areas for each of Minkowski sums are: $MSum(triangle_t, circle_r) = \frac{t^2\sqrt{3}}{4} + 3tr + \pi r^2$, $MSum(square_c, circle_r) = c^2 + 4cr + \pi r^2$, and $MSum(hexagon_s, circle_r) = s^2\frac{3\sqrt{3}}{2} + 6sr + \pi r^2$. After substitutions we find $MSum(triangle_t, circle_r) \approx s^2\frac{3\sqrt{3}}{2} + 7.35sr + \pi r^2$, and $MSum(square_c, circle_r) \approx s^2\frac{3\sqrt{3}}{2} + 6.45sr + \pi r^2$. Comparing all three equations, we find $MSum(hexagon_s, circle_r) < MSum(square_c, circle_r) < MSum(triangle_t, circle_r)$. This result means that for circular regions the Minkowski sum of a hexagon is always less than the Minkowski sum of a square and the Minkowski sum of a triangle. Substituting this result into the equation for $E_s(q)$, we finally find $E_{hexagon}(circle_r) < E_{square}(circle_r) < E_{triangle}(circle_r)$. The expected number of partitions intersected by a circular query in a hexagonal grid is always less than the one in a square grid and a triangular grid. This result simply tells us that if the sole or dominant types of the queries in a spatial database have circular shapes, then hexagonal partitioning of the data space will give better results in terms of the number of the page accesses, i.e., I/O operations, occurred during query.

Indeed, we will now generalize the superiority of hexagonal partitioning over other possible approaches and prove that hexagonal partitioning is optimal, among all non-overlapping partitioning techniques of equal area convex regions, with respect to the number partitions retrieved as a result of a circular query of any size.

Lemma 1. *For a convex polygon with a perimeter of length M, the Minkowski Sum with respect to circular regions of radius r is:*

$$MSum(polygon, circle_r) = area(polygon) + M \cdot r + \pi r^2 \tag{1}$$

Proof. For a convex polygon of n sides, the $MSum$ region with respect to a circle of radius r contains the polygon itself, plus n rectangles with sides r and the corresponding side of the polygon, plus n pies with an angle of Γ_i, for $1 \leq i \leq n$. The area of the n rectangles sum up to $M \cdot r$, where M is the perimeter of the polygon. Each Γ_i is equal to $180 - \Theta_i$ degrees, where Θ_i is the corresponding angle in the polygon. Since the summation of all these n angles of pies is 360 degrees, $\sum_1^n (180 - \Theta_i) = 180n - 180(n-2) = 360$, they sum up to a full circle.

Lemma 2. *Minimizing the perimeter of the shape that is used for partitioning also minimizes the expected number of partitions intersecting the circular query.*

Proof. In Equation 1, since r does not change with partitioning and $area(polygon) = 1/p$, the only value that makes a difference is the perimeter M that is used in the pages. This is true for all possible r values that a circular query can take.

Minimizing the perimeter of each partition minimizes the total boundary length of the partitioning. The total boundary of the partitions is defined as the length of the boundaries that are used to divide the data-space into partitions [3]. If each non-overlapping partition uses smaller perimeters to cover larger areas, the total lengths of boundaries of these partitions are also minimized [3]. What is the optimal way of minimizing the perimeter and the total boundary? Recently, it has been proved that any partitioning of the plane into regions of equal area has perimeter at least that of the regular hexagonal honeycomb tiling [5]. This has been known as the classical honeycomb observation and has been the motivation for numerous interesting applications. The surprising geometry of the beehives is the best that could be done for their major purpose. In 1743, Colin Mac Laurin summarizes the nice property of the beehives as follows: "The geometry of the beehive supports *least wax* for containing *the same quantity of honey*, and which has at the same time a very remarkable regularity and beauty, connected of necessity with its frugality" [6].

By Lemma 1 and 2 we conclude that a partitioning that minimizes the total boundary also minimizes the expected number of partitions retrieved as a result of a circular query. Since hexagon partitioning minimizes the total boundary compared to all possible equal area partitioning techniques, it also minimizes the expected number of partitions retrieved as a result of a circular query. Hence, we have the following theorem.

Theorem 1. *Hexagonal partitioning is I/O optimal for circular queries (among all non-overlapping partitioning techniques using equal area convex regions).*

3.2 I/O Cost for Square Range Queries

In this section, we analyze the I/O cost for square range queries. We start with rectilinear square query analysis on the three basic partitioning discussed in this paper. Then, we analyze a more general class of square range queries where the sides of the range query do not have to be parallel to the axes. We establish that hexagonal partitioning has better average performance than square grid for the general class of square range queries.

Rectilinear Square Range Queries For the analysis of rectilinear square range queries, we first compute the Minkowski sums of three shapes and a rectilinear square region. Similar to the previous analysis we find the Minkowski

Sum values for each shape as [3]:

$$Msum(square_c, rectilinear_a) = \frac{3\sqrt{3}}{2}s^2 + 2\sqrt{\frac{3\sqrt{3}}{2}}as + a^2,$$

$$MSum(hexagon_s, rectilinear_a) = \frac{3\sqrt{3}}{2}s^2 + (2+\sqrt{3})as + a^2,$$

$$MSum(triangle_t, rectilinear_a) = \frac{3\sqrt{3}}{2}s^2 + (\frac{3\sqrt{2}}{2} + \sqrt{6})as + a^2.$$

Comparing these three equations, we find $MSum(square_c, rectilinear_a) < MSum(hexagon_s, rectilinear_a) < MSum(triangle_t, rectilinear_a)$, and therefore,
$E_{square}(rectilinear_a) < E_{hexagon}(rectilinear_a) < E_{triangle}(rectilinear_a)$. We conclude that for rectilinear square queries, the expected number of intersected partitions in a square grid is less than the one in a hexagonal grid and a triangular grid. The technical report of this paper contains a detailed analysis of this case and also a similar analysis of the rectilinear rectangular range queries [3].

General Square Range Queries In the previous section, we focused on *rectilinear* square range queries, but queries can take any orientation and are not restricted to have sides parallel to the axes, e.g., as in diamond queries. In this section, we analyze *general* square queries with any *orientation*. We will compare the hexagonal and square partitioning by starting to compute the Minkowski Sums for square and hexagon tiles and general square region. The square region has an angle α with the $x - axis$. The examples of special cases include the diamond query, i.e., $\alpha = \pi/4$, and the rectilinear square query, i.e., $\alpha = 0$. In this section, for simplicity in $MSum$, we assume that the area of each partition is 1, i.e., $c^2 = s^2\frac{3\sqrt{3}}{2} = 1$. Figure 3 illustrates the case when $\alpha = \pi/4$. Because of the symmetric property of $MSum$, i.e., $MSum(hexagon_s, square_a) = MSum(square_a, hexagon_s)$, for simplicity we illustrate $MSum(square_a, hexagon_s)$ in Figure 3.

The $MSum$ of a square page with respect to a square query with an angle of α with the $x-$axis is:

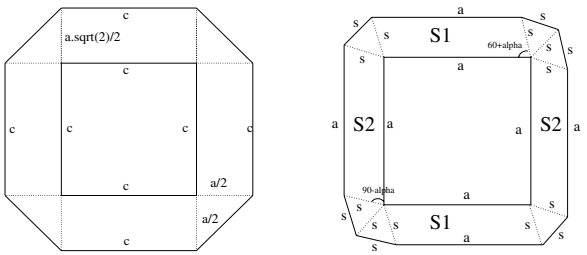
$$MSum(square_1, square_a) = 1 + a^2 + 2\sqrt{2}a \cdot sin(\pi/4 + \alpha) \quad (2)$$

where $0 \leq \alpha \leq \frac{\pi}{4}$. Similarly, the $MSum$ of a hexagon with respect to a square with a rotation angle of α, is computed as $MSum(hexagon, square_a) = a^2 + 1 + 2(S_1 + S_2)$, where $S_1 = as \cdot sin(\frac{\pi}{3} + \alpha)$ and $S_2 = as \cdot sin(\frac{\pi}{2} - \alpha)$. Therefore,

$$MSum(hexagon, square_a) = a^2 + 1 + 2as(sin[\frac{\pi}{3} + \alpha] + sin[\frac{\pi}{2} - \alpha]) \quad (3)$$

where $0 \leq \alpha \leq \frac{\pi}{6}$.

The reason that the rotation angle α varies between 0 and $\pi/6$ in the hexagon and between 0 and $\pi/4$ in the square is that the symmetry is captured within



(a) Square-Square (b) Square-Hexagon

Fig. 3. Minkowski Sums w.r.t. Squares (with any orientation α)

these angles. There is no need to compute the other angles, because rotating an angle of β not in this range gives the same result as rotating with an angle of $\alpha = \beta \bmod \pi/6$ for a hexagon and $\alpha = \beta \bmod \pi/4$ for a square. For majority of the angles which correspond to different query types, including the diamond query which represents similarity in L_1 metric, hexagonal partitioning achieves better performance results by requiring fewer I/O accesses. The technical report [3] has more discussion of the performance behavior of both partitioning under various angles.

Since we have the general formula for the square query specified with a center and an angle, we can compute the expected number of partitions intersected by such queries for both square and hexagonal grid. By integrating all such possible queries and computing the expected number by taking the uniform average, we can compute the expected $MSum$, E_{MSum}, of each technique. As shown before, the comparison of $MSum$s will give the comparison of techniques with respect to the expected number of partitions intersected by a random square query. Integrating Equations 2 and 3 for all possible values of α, we compute the expected $MSum$ for each partition as. For the case of square,

$$E_{MSum_{square}}(square_a) = \frac{4}{\pi}\int_{\pi/4}^{\pi/2}(MSum(square_1, square_a)$$
$$= a^2 + 1 + \frac{8\sqrt{2}a}{\pi}\int_{\pi/4}^{\pi/2} sin\alpha d\alpha,$$

$$E_{MSum_{square}}(square_a) = 1 + a^2 + \frac{8a}{\pi}. \tag{4}$$

Similarly, for hexagon,

$$E_{MSum_{hexagon}}(square_a) = 1 + a^2 + \frac{2as \cdot 6}{\pi}\int_0^{\pi/6}[sin(\pi/3+\alpha) + sin(\pi/4-\alpha)]d\alpha.$$

$$E_{MSum_{hexagon}}(square_a) = 1 + a^2 + \frac{12as}{\pi} \approx 1 + a^2 + \frac{7.44a}{\pi} \quad (5)$$

where $s \approx 0.62$ since we assumed the area of the hexagon as 1, i.e., $s^2 \frac{3\sqrt{3}}{2} = 1$.

Comparing Equations 4 and 5, we find $E_{MSum_{hexagon}}(square_a) < E_{MSum_{square}}(square_a)$. Hence, we conclude that the hexagonal partitioning minimizes the expected number of partitions intersected by a square query with any orientation α, compared to the square grid partitioning. The analysis can be easily extended for rectangular queries.

4 Retrieval and Storage of Hexagonal Partitioning

We have shown that the hexagonal partitioning is effective for range queries, and hence can be used as an effective alternative for regular grid partitioning. Traditional retrieval methods developed for single disk and single processor environments may be ineffective for the storage and retrieval of spatial data in multiprocessor and multiple disk environments. It is essential to develop techniques that are optimized for such environments. In this section, we discuss multi-disk organization for hexagonal partitioning, i.e., declustering of buckets to multiple disks and clustering each bucket within each disk. For lack of space, here we just describe how to develop a simple hash function for regular hexagonal partitioning, and we refer the technical report of this paper for a complete discussion of declustering and clustering functions. In particular, it discusses how techniques for rectangular grid partitioning can be adapted for hexagonal partitioning. It also proves that the number of devices needed for optimal declustering of hexagonal partitions is less than the rectangular grid partitions [3].

The hashing function finds the corresponding hexagonal partition of a given data point and is needed for both declustering and clustering purposes. Hash functions for rectangular partitioning are very simple and well-known. We will use rectangular hashing in the development of the hexagonal hashing. We divide the data space logically into a regular grid of rectangles with sides (s, h), where s is the side length of the hexagonal partitions and $h = \frac{s\sqrt{3}}{2}$ (See Figure 4). A hexagonal partition is defined by $H(i, j)$, where i is the row number and j is the column number of the partition. Similarly, a (logical) rectangular cell is defined by $G(i, j)$. Obviously, the number of rectangular cells is more than the number of hexagonal cells. Depending on the location, some of the cells in this regular grid fall entirely into a single hexagonal partition, and some fall into two hexagonal partitions. For example, in Figure 4, $G(0, 0)$ is entirely in $H(0, 0)$. Therefore, if a point is found to be in $G(0, 0)$, it is also in $H(0, 0)$. On the other hand $G(1, 1)$ falls mostly in $H(1, 1)$ but also in $H(0, 0)$. Each grid cell can be mapped to one or two hexagonal partitions. Therefore, given a data point we can hash the point using hashing on regular grid and find the corresponding hexagon(s). If there are two hexagons an additional simple check (whether the point is in the first hexagon) is needed to identify the hexagon where the point is located. The different cases for mapping of rectangular grid cells to hexagonal partitions are

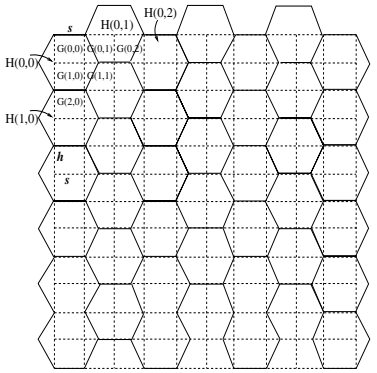


Fig. 4. Hexagonal hashing using grid

$G(2m, 3n) \subseteq H(m, 2n)$
$G(2m, 3n+1) \subseteq H(m, 2n) \cup H(m, 2n+1)$
$G(2m, 3n+2) \subseteq H(m, 2n+1) \cup H(m, 2n+2)$
$G(2m+1, 3n) \subseteq H(m, 2n)$
$G(2m+1, 3n+1) \subseteq H(m, 2n) \cup H(m+1, 2n+1)$
$G(2m+1, 3n+2) \subseteq H(m+1, 2n+1) \cup H(m, 2n+2)$

Fig. 5. Mapping of rectangular grid cells to hexagonal partitions

shown in Figure 5. A grid cell $G(i, j)$ is mapped to hexagonal partitions with a simple analysis on the value of (i mod 2) and (j mod 3). For example, grid cell $G(2m, 3n)$, i.e., $i \bmod 2 = 0$ and $j \bmod 3 = 0$, is mapped to $H(m, 2n)$. Given a point in $G(2m, 3n)$, e.g., $m = 2$ and $n = 1$ so $G(4, 3)$, it is located only in $H(m, 2n)$ (see Figure 5), e.g., $H(2, 2)$.

5 Discussion

In this paper, we explored optimal partitioning techniques for different types of queries on spatial data sets. We focused on partitioning techniques that tile the data space without holes and overlaps, and therefore have simple hashing schemes. We discussed the possible cases and computed the expected number of pages retrieved for circular and rectangular queries. We showed that hexagonal partitioning has optimal I/O cost for circular queries over all possible non-overlapping partitioning techniques that use convex regions. We also show that hexagonal partitioning has less I/O cost than the traditional grid for the general class of square queries. It is, however, interesting to note that for the special case of rectilinear queries, the traditional grid partitioning provides superior performance. This could be explained due to the symmetric relationship between the rectilinear square query and the rectilinear square page. This may also indicate why for traditional relational database applications only rectangular grid partitioning were considered [1,7]. In a relational database, a select operation specifies a range in each dimension or attribute which corresponds to a rectilinear rectangle. Novel spatial applications need more general query structures with queries of any orientations. Our results indicate that for such applications, a hexagonal partitioning of the space should be used. The hexagonal partitioning is shown to be effective for circular and rectangular queries. It can be used as an effective alternative for regular grid partitioning with no major changes

on the existing algorithms. For instance, a widely used application of regular grid partitioning is declustering where non-overlapping partitions are created and distributed to multiple disks for efficient I/O. We showed how to adapt the techniques that were developed for regular grid partitioning to hexagonal partitioning and develop techniques for storage and retrieval of hexagonal partitioning in multi-disk environments. The details of these techniques can be found in the technical report of this paper [3]. The techniques developed in this paper consider the objectives that are crucial for multi-disk searching: i) minimizing the number of page accesses during the execution of the query, ii) maximizing the I/O parallelism, iii) minimizing the disk-arm movement (seek time). We plan to extend our techniques to support efficient retrieval of clustered data sets also in the presence of frequent updates.

References

1. H. C. Du and J. S. Sobolewski. Disk allocation for cartesian product files on multiple-disk systems. *ACM Transactions of Database Systems*, 7(1):82–101, March 1982. 890, 899
2. H. Ferhatosmanoglu, D. Agrawal, and A. El Abbadi. Concentric hyperspaces and disk allocation for fast parallel range searching. In *Proc. Int. Conf. Data Engineering*, pages 608–615, Sydney, Australia, March 1999. 890
3. H. Ferhatosmanoglu, D. Agrawal, and A. El Abbadi. Optimal partitioning for spatial data. Technical report, Comp. Sci. Dept., UC, Santa Barbara, December 2000. 892, 895, 896, 897, 898, 900
4. V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998. 889
5. Thomas C. Hales. The honeycomb conjecture. available at http://xxx.lanl.gov/abs/math.MG/9906042, June 1999. 895
6. Thomas C. Hales. Historical background on hexagonal honeycomb. http://www.math.lsa.umich.edu/ hales/countdown/honey/hexagonHistory.html, March 2000. 895
7. S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Cyclic allocation of two-dimensional data. In *International Conference on Data Engineering*, pages 94–101, Orlando, Florida, Feb 1998. 890, 899
8. H. Samet. *The Design and Analysis of Spatial Structures*. Addison Wesley Publishing Company, Inc., Massachusetts, 1989. 889
9. R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 194–205, New York City, New York, August 1998. 890, 892