Searching the 2DR-tree

Wendy Osborn* and Ken Barker†
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, AB
T2N 1N4 Canada

Abstract

The 2DR-tree is a novel approach for accessing spatial data, which uses nodes that are the same dimensionality as the data space. Therefore, all relationships between objects are preserved and different binary search strategies are supported. This paper presents the 2DR-tree binary search strategy. Validity rules ensure that each node preserves all spatial relationships. A performance evaluation shows the advantages of the 2DR-tree binary search strategy.

1 Introduction

Many applications rely on spatial data. Some include geographical information systems (GIS) and bioinformatics databases. For example, Alberta Energy and Utilities Board [AEUB04] maintains a GIS on the provincial geological survey for energy and mineral resources. The Research Collaboratory for Structural Bioinformatics [Bio04] maintains a repository of three-dimensional protein structures. These "non-standard" databases contain large amounts of data that is represented in multidimensional space. An important issue for large spatial data sets is efficient retrieval of specific objects or subsets of spatial objects.

No n-dimensional to one-dimensional mapping of spatial data that preserves all spatial relationships exists [GG98]. However, most multidimensional indexing structures

*This research funded by an NSERC Discovery Grant.
†Contact author.

proposed in the literature extend hierarchical data structures originally proposed for one-dimensional data. These structures lead to several problems including: unnecessary searching, unnecessary entry checking in nodes, and superlinear growth. Attempts to map n-dimensional data to a one-dimensional sequence of values do not maintain all spatial relationships between objects.

A new hierarchical spatial access method, the nDR-tree, is proposed that fits the existing data by using nodes that are the same dimensionality. The two-dimensional version, the 2DR-tree, is presented here which uses two-dimensional nodes to index two-dimensional data. The minimum bounding rectangles in each node are organized according to a "validity rule" that preserves spatial relationships. This provides support for different types of binary searches, including: region, point, containment, enclosure, and equality.

The remainder of the paper is organized as follows. Section 2 presents related work in the area of spatial access methods. Section 3 introduces the 2DR-tree and defines key concepts. Section 4 presents the 2DR-tree binary search strategy. Section 5 shows an example binary search. Section 6 presents performance evaluations region and point searches. Finally, Section 7 presents conclusions and future research directions.

2 Related Work

Many approaches for indexing spatial data are proposed in the literature. In addition, several surveys [GG98, Sam90, RSV01] have been published. Spatial access methods are classified into three categories [GG98]: main memory access methods [FB74, Ben75, Ben79], point access methods [FBF77, Ben79, Rob81, HSW89, NHS81, NHS84, Hin85] and spatial access methods [Gut84, BKSS90, BKK96, AWYE97, CPZ97, KF94, SRF87, OM84, Ore89, Bay97, SK96, Kou00].

Main memory access methods extend the Binary Search Tree (BST) to handle sets of attribute values (i.e. points). These methods recursively partition the space based on different attribute values. The quadtree [FB74] recursively partitions the space into quadrants using the dimension values stored in each node. The k-d tree [Ben75, Ben79]

performs an alternating recursive binary partition using the assigned dimension value stored in each node. Both structures are easy to implement. However, both have limitations including: point data, skewing in the presence of dynamic data, and lack of secondary storage.

Point access methods (PAMs) are space-partitioning techniques that take secondary storage and, to some extent, height balancing into consideration. PAMs are classified into hierarchical and non-hierarchical methods. Hierarchical PAMs recursively partition the space using a tree structure, but are organized for efficient access from secondary storage. Hierarchical PAMs include the adaptive k-d tree [FBF77, Ben79] and the k-d-B tree [Rob81] which also takes balancing into consideration. Non-hierarchical (i.e., grid file) PAMS [NHS81, NHS84] extend the directory structure from extendible hashing to handle point data. Although grid files support constant-time searching for point queries, they experience superlinear growth with many empty buckets.

Spatial access methods (SAMs) provide uniform access to both point and object data. In addition, they remain height-balanced when dynamic data is present. SAMs are classified [GG98] into approximation methods, clipping methods, and mapping methods. Approximation methods store a hierarchy of minimum bounding rectangles that represent both the objects and the space they occupy. Since the space is not partitioned, minimum bounding rectangles can overlap. Many approximation SAMs are proposed, including the R-tree [Gut84], the R^*-tree [BKSS90], X-tree [BKK96], the M-tree [CPZ97], and the S-tree [AWYE97]. Clipping methods, such as the R^+-tree [SRF87], partition an object to be stored in non-overlapping minimum bounding rectangles. Mapping methods provide a one-dimensional ordering for n-dimensional objects, which allows storage and access using an existing one-dimensional access method such as a B-tree [BM74]. Approaches using mapping include Z-ordering [OM84, Ore89], the Hilbert R-tree [KF94], the Universal B-tree [Bay97], and the Filter tree [SK96, Kou00].

Günther and Gaede [GG98] state that no n-dimensional to one-dimensional mapping of spatial data exists that preserves all spatial relationships. A limitation to most proposed access methods is their one-dimensional structure that requires forcing n-dimensional data to fit them. This one-dimensional limitation leads to inefficient

searching, both within the structure and within a node itself because the only option is a linear search. Grid files are n-dimensional in structure and support constant-time search but have superlinear growth and possibly many empty buckets. Mapping methods do not preserve all spatial relationships and therefore have the same problems with searching as other hierarchical SAMS.

3 The 2DR-tree

The 2DR-tree is a hierarchical spatial data structure where each node is two-dimensional. A minimum bounding rectangle is stored in an appropriate location with respect to the spatial location of all other objects in the node. Using two-dimensional nodes allows minimum bounding rectangles to be placed so that the spatial relationships are preserved.

A minimum bounding rectangle defines the minimum extent along both the x-axis and the y-axis that encompass either an object or a collection of minimum bounding rectangles. The extent along an axis is defined using lower and upper co-ordinate values of an object or a collection of objects. The centroid of a minimum bounding rectangle is the co-ordinate pair (i, j) of its centre.

For each node N, X is the number of index locations along the x-axis, and Y is the number of index locations along the y-axis. The order of N is defined as $O = X * Y$. All nodes in a 2DR-tree have the same order. Therefore, the order of a 2DR-tree is the order of its nodes. $N_{(i,j)}$ denotes location (i, j) in node N. Each location stores:

$$(MBR_{(i,j)}, ptr_{(i,j)})$$

where $MBR_{(i,j)}$ is a minimum bounding rectangle and $ptr_{(i,j)}$ is a pointer. In a leaf node, $MBR_{(i,j)}$ encloses an object and $ptr_{(i,j)}$ references the object on secondary storage. In a non-leaf node, $MBR_{(i,j)}$ encloses all minimum bounding rectangles in the subtree referenced by $ptr_{(i,j)}$.

A node region (lx, hx, ly, hy) is a two-dimensional subset of (x, y) index locations in a node. The index values lx and hx are the lower and upper bounds of the node region

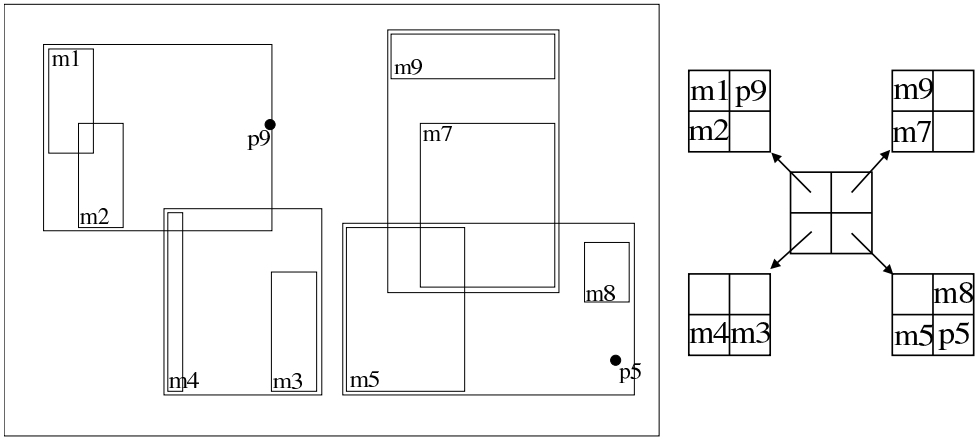


Figure 1: Order 4*4 2DR-tree

along the x-axis. The index values ly and hy are the lower and upper bounds of the node region along the y-axis. A node object set is the set of objects in a node. A node space is the region occupied by a node objects set. This is equal to the area of the minimum bounding rectangle that encloses a node object set.

Figure 1 shows an order 2*2 2DR-tree that preserves all spatial relationships for the given data set, which is a subset of the running example from Günther and Gaede [GG98]. Beginning with the leaf node descending from location $(0, 0)$ in the root, the centroid for m3 is located southeast of the centroid so m3 is located east of m4 in the node (m4,m3). In node (m5,p5,p8), p5 is located southeast of the centroid for m5, and the centroid for m8 is located northeast of the centroid for m5 and northwest of p5. Therefore, p5 is stored east of m5 while m8 is stored northeast of m5 and north of p5. The remaining two leaf nodes, (m7,m9) and (m2,m1,p9) preserve the spatial relationships between the objects each contains. Spatial relationships between minimum bounding rectangles in the root node are also preserved.

3.1 Node Validity

To employ different binary searching strategies, the spatial relationships between minimum bounding rectangles in each node must be preserved. For each location $N_{(i,j)}, i = 0 \ldots (X-1), j = 0 \ldots (Y-1)$ in node N, if $N_{(i,j)}$ contains a minimum bounding rectangle $MBR_{(i,j)}$,

- Location $N_{(k,l)}, k = (i+1) \ldots (X-1), l = 0 \ldots j$ contains $MBR_{(k,l)}$ whose centroid is southeast of the centroid for $MBR_{(i,j)}$,
- Location $N_{(k,l)}, k = (i+1) \ldots (X-1), l = (j+1) \ldots (Y-1)$ contains $MBR_{(k,l)}$ whose centroid is northeast of the centroid for $MBR_{(i,j)}$, and
- Location $N_{(k,l)}, k = 0 \ldots i, l = (j+1) \ldots (Y-1)$ contains $MBR_{(k,l)}$ whose centroid is northwest of the centroid for $MBR_{(i,j)}$.

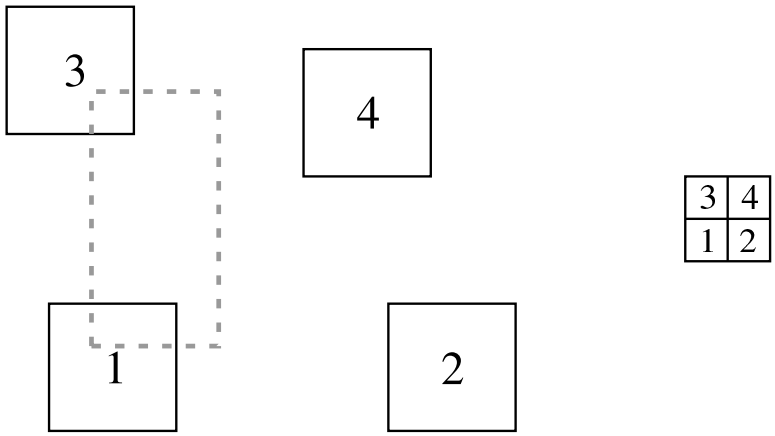
When constructing the 2DR-tree, a validity testing strategy ensures that each minimum bounding rectangle is in a proper location with respect to all other minimum bounding rectangles in the node.

4 Binary Search

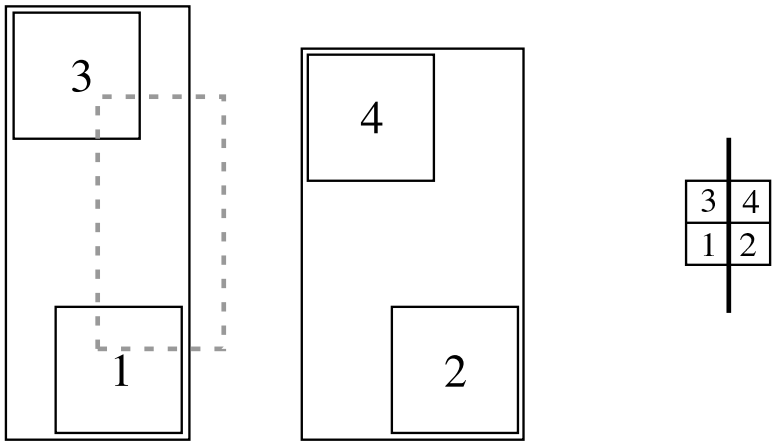
This section presents the 2DR-tree binary search strategy. The strategy works for region, point, containment, enclosure, and equality spatial queries. The following discusses the binary search strategy for region queries.

The search begins by performing a recursive binary partition of the node region in the root. First, the node region $(lx_{rt}, hx_{rt}, ly_{rt}, hy_{rt})$, which contains all locations in the root, is identified. Second, the partitioning dimension is identified. The dimension with the longest range of index values in the node region is chosen. Third, the node region is partitioned through the midpoint of the chosen dimension. A partition through the x-axis at midpoint cx_{rt} produces $(lx_{rt}, cx_{rt} - 1, ly_{rt}, hy_{rt})$ and $(cx_{rt}, hx_{rt}, ly_{rt}, hy_{rt})$. A partition through the y-axis through the midpoint cy_{rt} produces $(lx_{rt}, hx_{rt}, ly_{rt}, cy_{rt} - 1)$ and $(lx_{rt}, hx_{rt}, cy_{rt}, hy_{rt})$. For each new node region, a minimum bounding rectangle is then created which includes all objects in it.

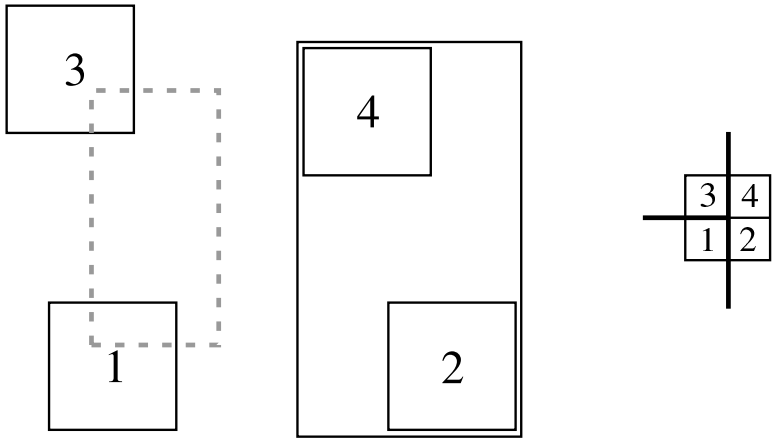
Each minimum bounding rectangle is then tested for overlap with the search region. If overlap occurs, the node region undergoes further binary partitioning until no more overlap occurs, the node region contains no minimum bounding rectangles, or the node region contains one location with a minimum bounding rectangle. If one minimum bounding rectangle is reached, it is tested for overlap with the search region. If overlap



(a) Node and Data



(b) X-Partition



(c) Y-Partition

Figure 2: Recursive Binary Partitioning

occurs, the search continues in the corresponding subtree.

If a leaf node is reached, the same binary partition is performed. If a node region containing one object is reached, the object is tested for overlap with the search region. If the object overlaps, it is returned as part of the result.

Figure 2 shows the recursive binary partition of a node. In Figure 2(a) the node region that contains all locations is $(0, 1, 0, 1)$. Figure 2(b) shows a X-partition of the node region and subsequent minimum bounding rectangle creation. The query region, indicated with a grey dashed rectangle, overlaps the minimum bounding rectangle corresponding to the first node subregion but does not overlap the minimum bounding rectangle corresponding to the second node subregion. Figure 2(c) shows a Y-partition of this node subregion that leads to two more subregions, each with one minimum bounding rectangle that overlap the query region. If the node is a non-leaf node, the binary search continues in the corresponding subtree. If the node is a leaf node, the objects are returned as part of the result because both overlap the query region.

5 Example

This section demonstrates a 2DR-tree region binary search using the order 2*2 2DR-tree in Figure 3. In the example, subtrees are displayed using the objects it contains, and one or more levels of parentheses to indicate its height. For examples, (m4,m3) represents a leaf node (i.e.: height 0), while ((m4,m3),(m2,m1,p9)) represents a subtree of height 1, containing two minimum bounding rectangles that reference (m4,m3) and (m2,m1,p9), respectively.

The search begins by performing a X-partition of the node region $(0, 1, 0, 1)$ in the root, which produces $(0, 0, 0, 1)$ and $(1, 1, 0, 1)$. The node subregion $(0, 0, 0, 1)$ contains ((m4,m3),(m2,m1,p9)) and $(1, 1, 0, 1)$ contains ((m5,p5,m8),(m7,m9)). Minimum bounding rectangles are created for the subregions. Both overlap the query region. The binary partitioning continues first with $(0, 0, 0, 1)$. Since the Y-range is greater than the X-range, a Y-partition is performed to produce $(0, 0, 0, 0)$ and $(0, 0, 1, 1)$. The subregion $(0, 0, 0, 0)$ contains one minimum bounding rectangle which overlaps the query region.

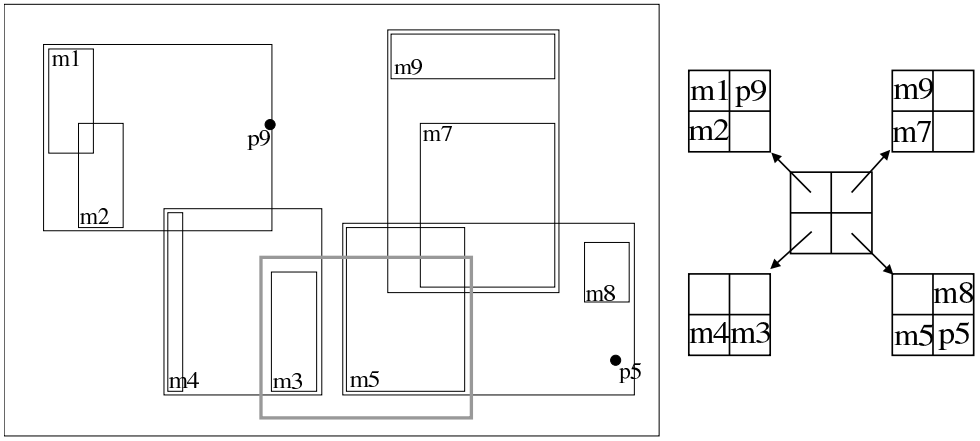


Figure 3: Region Query

The search continues in the node $(m4, m3)$. Again, since the X-range and Y-range are equal, a X-partition is performed to create $(0, 0, 0, 1)$ and $(1, 1, 0, 1)$. The subregion $(0, 0, 0, 1)$ contains $(m4)$ and $(1, 1, 0, 1)$ contains $(m3)$. Although both subregions contain one minimum bounding rectangle each, they both contain two locations, so the minimum bounding rectangles are created. The minimum bounding rectangle for $(0, 0, 0, 1)$ does not overlap the query region, so it is not partitioned further. The minimum bounding rectangle for $(1, 1, 0, 1)$ overlaps the query region. A Y-partition is performed to create $(1, 1, 0, 0)$ and $(1, 1, 1, 1)$. The subregion $(1, 1, 0, 0)$ contains one minimum bounding rectangle (m4) that overlaps the query region and is returned as part of the result. The subregion $(1, 1, 1, 1)$ contains no minimum bounding rectangles.

The search has finished with node $(m4, m3)$ so it backtracks to the next subregion in the root $(0, 0, 1, 1)$. The subregion contains one minimum bounding rectangle, ((m2,m1,p9)) that does not overlap the query region. The search moves to the minimum bounding rectangle for $(1, 1, 0, 1)$, which overlaps the query region. A Y-partition produces $(1, 1, 0, 0)$ and $(1, 1, 1, 1)$. Each contains a minimum bounding rectangle that overlaps the query region. The search continues first in the subtree corresponding to $(1, 1, 0, 0)$, and through a recursive binary partition of (m5,p5,m8) locates m5 that overlaps the query region. Backtracking to the root, the search continues in the node $(m7, m9)$ and performs a recursive binary partition to locate m7 as the final qualifying object.

6 Evaluation

This section presents the methodology and results of a 2DR-tree binary search performance evaluation. The evaluation methodology is presented first followed by a discussion of the experimental results.

6.1 Methodology

The rationale for the binary search performance evaluation is to observe its behaviour on trees that vary in the number of objects data distributions. The 2DR-tree is implemented using Java on Dell Precision WorkStation 530 MT running Windows 2000. The object and query sets are stored in databases managed by IBM DB2.

The performance evaluation uses several object sets to create the trees used for the searches. UNIF100, UNIF500, UNIF1000, and UNIF10000 contain 100, 500, 1000, and 10000 squares, respectively. Each square covers approximately 1% of the "space". Each set of objects is uniformly distributed. Each set covers approximately 90-100% of the space without considering overlap. Each set covers approximately 67-75% of the space with between 51-53% overlap. SKEW500 contains 500 squares with an exponential (i.e.: skewed) distribution. This set was generated separately, but the squares are the same dimension, and achieve the same non-overlap coverage as UNIF500, although more overlap exists due to the distribution.

The binary search performance evaluation also uses several query sets. QU100, QU500, QU1000, and QU10000 each contain 1000 uniformly distributed rectangles. Each rectangle covers 5-10% of the "space" where the extent of the space for each set corresponds to the extent used for the matching uniform data set (e.g. rectangles in QU100 cover 5-10% of the extent of UNIF100). P100, P500, P1000, and P10000 contain 1000 uniformly distributed points. The points in each set correspond to the centroids for each rectangle in the corresponding query set. Although each set contains 1000 rectangles or points, only the first 100 are used in a test run.

For each test run, 100 queries are executed on 1000 2DR-trees. Each set of 1000 trees are generated using random sorts of different object set. The binary search performance

is evaluated for both region and point queries. Each are evaluated using two sets of test runs. The first set of test runs execute queries on trees that vary in the number of objects. The second executes queries on trees that vary in the data distribution.

The following averages are recorded for each test run: the tree height, the number of seeks per search, the number of objects retrieved per search, and a ratio of the number of seeks per search over the number of objects retrieved per search. The latter determines the number of seeks required to locate an object. The metric used to judge performance is a comparison of the number of seeks required to locate an object vs. the tree height. In a B^+-tree, the length of the search path is equal to the height of the tree. R-tree structures including the 2DR-tree support retrieval of multiple objects and cannot guarantee a one-path search. This is true for region queries, point queries, and locating an individual object because many minimum bounding rectangles can overlap at a single point or region. Therefore the search may have to backtrack part of all of the way up the tree and search several paths to locate all objects that overlap a point or region. Ideally, the search locates an object and backtracks one level to locate another object. Pessimistically, the search must search several different paths - all of which originate at the root - to locate one object. Therefore, our performance measure is the average number of seeks required to locate each object versus the height of the tree. If the average number of seeks per object is less than the height, this shows that searches perform well because the search does not have to locate each object starting from the root. Search performance deteriorates when the average number of seeks per object is greater than the height of the tree because multiple paths are searched to locate each object.

6.2 Results and Discussion

The results of the performance evaluation are considered in this section. The results for the region queries are presented first, followed by the results for the point queries.

#Objects	Height	#Seeks/Srch	#Found/Srch	#Seeks/#Found
100	8.10	16.38	2.56	6.40
500	12.31	40.94	6.90	5.93
1000	14.19	60.98	10.97	5.56
10000	21.00	268.27	73.27	3.66

Figure 4: Average from Region Binary Search with Varying Object Set Size

6.2.1 Region Binary Search

Figure 4 shows the results for the first set of test runs. Results show that the number of seeks required per object decreases as the number of objects increase. This is very significant because the height also increases as the number of objects increases. In all cases, the number of seeks per object is lower than the height. In addition, the more objects inserted, the better the 2DR-tree is able to cluster them so that fewer paths need to be searched to find them.

Row 1 shows the average height for 1000 trees that each contain 100 objects. Also displayed are the average number of seeks per search, average number of objects found per search and the average number of seeks required to locate an object over all 100 queries on all 1000 trees. The average number of seeks required to locate an object is 6.40, which is 80% of the tree height of 8.10. For row 4, where the trees contain 10000 the average number of seeks per object is 3.66, which is only 17% of the tree height of 21. Therefore, the search does is not required to backtrack all the way to the root to locate each object.

Figure 5 shows the results for the second set of test runs. Once again the results show that the number of seeks required per object is less than the tree height. It is lower for exponential distribution because the actual search results vary widely - many

Distribution	Height	#Seeks/Srch	#Found/Srch	#Seeks/#Found
Uniform	12.31	40.94	6.90	5.93
Exponential	12.31	29.11	8.28	3.52

Figure 5: Average from Region Binary Search with Varying Distribution

#Objects	Height	#Seeks/Srch	#Found/Srch	#Seeks/#Found
100	8.10	12.96	1.06	12.23
500	12.31	26.12	1.12	23.32
1000	14.19	34.51	1.10	31.37
10000	21.00	96.04	1.32	72.76

Figure 6: Average from Point Binary Search with Varying Object Set Size

queries retrieve nothing while some retrieve many objects. Therefore, it appears that few nodes are accessed when nothing is retrieved. For uniform distribution, the range of the number of objects retrieved is very narrow in comparison to exponential distribution, with many queries returning objects.

6.2.2 Point Binary Search

Figure 6 shows the results for the first set of test runs. Results show that the number of seeks per object is greater than the height of the tree. The point queries are finding the wasted space in addition to any objects it overlaps because of the uniform distribution of the data.

In Row 1, the average number of seeks per object is 12.23, which is 150% of the tree height of 8.1 and shows that multiple paths are searched to locate one object. This deteriorates as the number of objects increases. In Row 4, the average number of seeks per object is 72.76, which is over 300% of the tree height. In this case, three paths that originate at the root are required to locate an object.

Figure 7 shows the results for the second set of test runs. Results show that point searching improves significantly for exponential distribution, which requires fewer seeks than the height of the tree to locate an object. Since the data is clustered more, a

Distribution	Height	#Seeks/Srch	#Found/Srch	#Seeks/#Found
Uniform	12.31	26.12	1.12	23.32
Exponential	12.31	15.15	1.40	10.82

Figure 7: Average from Point Binary Search with Varying Distribution

point is searching in less wasted space than in uniformly distributed data, and more importantly, search very little in areas not occupied by objects at all. Therefore, the 2DR-tree is not suited to point queries unless the data is clustered in some way.

7 Conclusion

This paper presents the 2DR-tree binary search strategy. The 2DR-tree is a spatial access method that preserves spatial relationships between all objects by using nodes that are the same dimension as the object set. Currently, the access method supports different types of searches including region, point, containment, enclosure, and equality.

Experimental results show that the 2DR-tree is ideal for exeucting region queries where the search region is between 5-10% of the search space. Region search performance improves with the number of objects in the tree. In addition, the region search performs well in both uniform and exponential data. Unfortunately, performance is poor for point queries. Distribution and overcoverage are a factor in this performance, since point queries still perform well when the distribution is exponential and overcoverage is lower. Since point queries are region queries with all corners equal, this suggests that as the region query decreases in size, the search performance decreases.

Future research directions include the following. The first is to extend the performance evaluation to consider varying sized region queries. The second is to compare the search performance of the 2DR-tree against other proposed spatial access methods. The third is to develop algorithms for other types of spatial searches, such as adjacency, nearest neighbour, and spatial join. Finally, extended the binary search strategy to work in multiple dimensions will be explored.

References

[AEUB04] Alberta Energy and Utilities Board. Alberta Geographical Survey GIS, http://www.ags.gov.ab.ca/gis/gis_and_mapping.shtml. Access Date May, 2004.

[AWYE97] C. Aggarwal, J. Wolf, P. Yu, and M. Epelman. The S-tree: an efficient index for multidimensional objects. In *Advances in Spatial Databases: 5th International Symposium on Spatial Databases SSD '97*, number 1262 in Lecture Notes in Computer Science, pages 350–73. Springer-Verlag, 1997.

[Bay97] R. Bayer. The universal B-tree for multidimensional indexing: general concepts. In *Worldwide Computing and Its Applications, WWCA '97*, number 1274 in Lecture Notes in Computer Science, pages 198–209. Springer-Verlag, 1997.

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[Ben79] J.L. Bentley. Multidimensional binary search trees in database applications. *IEEE Transacations on Software Engineering*, 5:333–340, 1979.

[Bio04] Research Collaboratory For Structural Bioinformatics. Protein data bank, http://www.rcsb.org/pdb. Access Date May, 2004.

[BKK96] S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, pages 28–39, 1996.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–31, 1990.

[BM74] R. Bayer and E.M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–188, 1974.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

[FB74] R.A. Finkel and J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[FBF77] J.H. Friedman, J.L Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–26, 1977.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[Hin85] K. Hinrichs. Implementation of the grid file: design concepts and experience. *BIT*, 25:569–92, 1985.

[HSW89] A. Henrich, H.-W. Six, and P. Widmayer. The LSD-tree: spatial access to multidimensional point and non-point objects. In *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, pages 45–53, 1989.

[KF94] I. Kamel and C. Faloutsos. Hilbert R-tree: an improved r-tree using fractals. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 500–9, 1994.

[Kou00] N. Koudas. Indexing support for spatial joins. *Data and Knowledge Engineering*, 34:99–124, 2000.

[NHS81] J. Nievergelt, H. Hintenberger, and K. Sevcik. The grid file: an adaptable symmetric multikey file structure. In *Proceedings of the Third ECI Conference*, number 123 in Lecture Notes in Computer Science, pages 236–51. Springer-Verlag, 1981.

[NHS84] J. Nievergelt, H. Hintenberger, and K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9:38–71, 1984.

[OM84] J Orenstein and T.H. Merrett. A class of data structures for associative searching. In *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 181–90, 1984.

[Ore89] J. Orenstein. Redundancy in spatial databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 294–305, 1989.

[Rob81] J.T. Robinson. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 10–18, 1981.

[RSV01] P. Rigaux, M. Scholl, and A. Voisard. *Spatial databases: with application to GIS*. Morgan-Kauffman, 2001.

[Sam90] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley, 1990.

[SK96] K.C. Sevcik and N. Koudas. Filter trees for managing spatial data over a range of size granularities. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, pages 16–27, 1996.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: a dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, 1987.