Proceedings of the 10th IASTED International Conference
Internet and Multimedia Systems and Applications
August 14–16, 2006, Honolulu, Hawaii, USA
ISBN Hardcopy: 0-88986-587-6 / CD: 0-88986-589-2

SEARCHING THROUGH SPATIAL RELATIONSHIPS USING THE 2DR-TREE

Wendy Osborn*
Department of Mathematics
and Computer Science
University of Lethbridge
4401 University Drive W
Lethbridge, Alberta, Canada
email: wendy.osborn@uleth.ca

Ken Barker
Department of Computer Science
University of Calgary
2400 University Drive NW
Calgary, Alberta, Canada
email: barker@cpsc.ucalgary.ca

ABSTRACT
The 2DR-tree is a novel approach for accessing spatial data, particularly when spatial relationships are defined among objects. The 2DR-tree 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. A preliminary performance evaluation identifies the advantages of the binary search strategy.

KEY WORDS
spatial access methods, multidimensional, hierarchical.

1 Introduction

Many applications rely on spatial data. Some include geographical information systems (GIS). For example, the Geological Survey of Canada [1] maintains a repository of spatial data for many geoscience applications. These "non-standard" databases contain objects that exist in multidimensional space. In addition, spatial relationships exist between objects. For example, resources in a geological survey are related (explicitly or implicitly) using directions such as north, northeast, southwest, etc.

An important issue in spatial databases is the efficient retrieval of one or more objects using a spatial access method (SAM). Existing SAMs extend hierarchical data structures designed for one-dimensional data. However, no n-dimensional to one-dimensional mapping of spatial data that preserves all spatial relationships exists [2]. Therefore, two objects in a specific spatial relationship may not maintain this relationship in the index. This leads to unnecessary searching because a linear search of an entire node is the only option.

A new hierarchical SAM called the 2DR-tree is proposed that fits the existing object space by using nodes of the same dimensionality. The 2DR-tree uses two-dimensional nodes for a two-dimensional object space. The minimum bounding rectangles (MBRs) in each node are organized using a validity rule that preserves all spatial relationships. Therefore, the 2DR-tree can exploit other search strategies, including a two-dimensional binary search. The 2DR-tree supports the region, point, containment, enclosure, and equality searches. It also has the potential to support spatial network data [3] and spatial reasoning strategies for direction relations [4].

We present the 2DR-tree binary search strategy. The remainder of the paper proceeds as follows. Section 2 presents related work. Section 3 presents the 2DR-tree and some key concepts. Section 4 defines a two-dimensional binary search and presents the search strategy. Section 5 presents an example search. Section 6 presents performance evaluations for region and point searches. Section 7 presents conclusions and future research directions.

2 Related Work

Many SAMs are proposed in the literature [5, 6, 7, 8, 9, 10, 11, 12]. An excellent survey on SAMs is presented in [2]. SAMs are based on the B^+-tree [13]. Therefore, SAMs remain height-balanced after insertions and deletions, and guarantee a minimum spatial utilization in its nodes.

The R-tree [5], R^*-tree [6], R^+-tree [7], and X-tree [8] maintain a hierarchy of approximations using MBRs. At the leaf level, an MBR defines the extent of an object in space. At each non-leaf level, an MBR contains all MBRs in the subtree that it references. A region search is performed by testing all MBRs in a node for overlap with the search region. For each MBR that overlaps, the search continues in the corresponding subtree. For each leaf node reached, all of its MBRs are tested for overlap with the search region. Point queries are performed similarly.

The Hilbert R-tree [9], Z-ordering [10], Universal B-tree [11], and Filter tree [12] map objects in n-dimensional space to a one-dimensional sequence of numbers and store them in a B^+-tree [13]. Region searches are performed by identifying the range of numbers that correspond to a region and locating all numbers in the index that fall within this range. An exception is the Hilbert R-tree, which stores MBRs and supports the searching strategy for approximation hierarchies.

*Research funded by an NSERC Discovery Grant.

Limitations of SAMs include the following. First, overlap and overcoverage of MBRs in approximation strategies lead to multiple path searching, including paths that lead to no objects that overlap the search region. Second, n-dimensional objects are indexed using traditional one-dimensional data structures. However, no n-dimensional to one-dimensional mapping of objects exists that preserves all spatial relationships between objects [2]. Therefore, two objects that have a specific spatial relationship may not maintain this relationship in the index. In approximation strategies, a linear search of an entire node is the only option. In mapping strategies all leaf nodes that are retrieved must be searched in their entirety to identify qualifying objects. This leads to inefficient searching within each node and in the structure overall.

3 The 2DR-tree

The 2DR-tree is a height-balanced, hierarchical spatial data structure that uses two-dimensional nodes. An MBR is stored in an appropriate location with respect to all other MBRs in the node. Using two-dimensional nodes allows spatial relationships to be preserved.

The spatial relationships supported in the 2DR-tree are north, northeast, east, southeast, south, southwest, west and northwest. A spatial relationship is defined between two objects using the centroids of their MBRs. The centroid of an MBR are the co-ordinates (i, j) of its centre. For example, MBR 1 is northeast of MBR 2 if the centroid of MBR 1 is northeast of the centroid of MBR 2. Relating objects using centroids instead of relating the objects themselves [4] significantly reduces the number of spatial relationships that must be handled in the 2DR-tree, but still provides sufficient support for spatial relationships.

For each node N, X is the number of index values along the x-axis, and Y is the number of index values along the y-axis. The order of N - the total number of locations in N to store an MBR - is defined as $O = X * Y$. All nodes in a 2DR-tree have the same order. 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 an MBR 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 MBRs in the subtree referenced by $ptr_{(i,j)}$.

A node region (lx, hx, ly, hy) is a two-dimensional subset of (x, y) locations in a node. The index values lx and hx are the lower and upper bounds of the node region 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 object set. This is equal to the MBR that encloses a node object set.

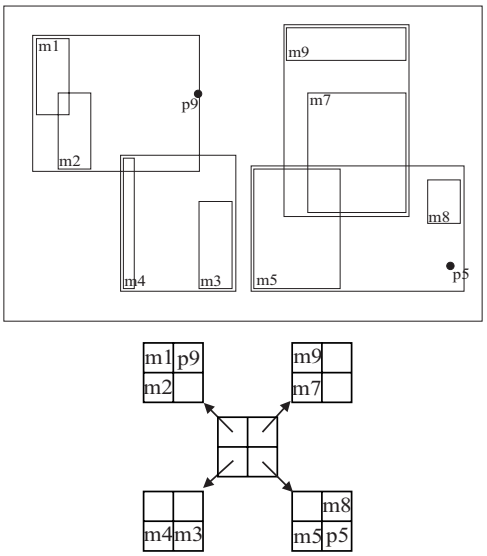


Figure 1. Order 2*2 2DR-tree

Figure 1 shows an order 2*2 2DR-tree that preserves all spatial relationships for the given objects (taken from [2]). In the leaf node (m4,m3), the centroid of m3 is located southeast from the centroid of m4, so m3 is located east of m4 in the node. 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 their objects. Spatial relationships between MBRs in the root are also preserved.

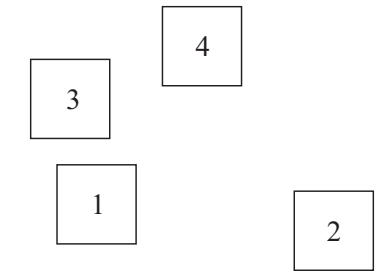
3.1 Node Validity

To employ different binary searching strategies, the spatial relationships between MBRs 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 an MBR $MBR_{(i,j)}$,

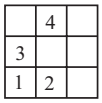
- Location $N_{(k,l)}, k = (i+1) \ldots (X-1), l = 0 \ldots j$ contains $MBR_{(k,l)}$ whose centroid is south, east or southeast of the centroid for $MBR_{(i,j)}$,
- Location $N_{(k,l)}, k = (i+1) \ldots (X-1), l = (j+1) \ldots (Y-1)$ contains $MBR_{(k,l)}$ whose centroid is northeast of the centroid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

A southwest test is not required because it is the inverse of the northeast test.

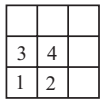
Figure 2 depicts the node validity for the set of MBRs in Figure 2(a). The node in Figure 2(b) is valid for the set



(a) Set of MBRs



(b) Valid



(c) Invalid

Figure 2. Node Validity

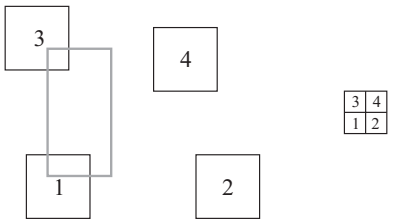
of MBRs. MBR 1 is in location (0,0). For MBR 2 to be in location (1,0), its centroid must be, and is, southwest of the centroid for MBR 1. Similarly, for MBR 3 to be in location (0,1), its centroid must be located northwest of the centroid for MBR 1. For MBR 4 to be in location (1,2), its centroid must be northeast of both MBRs 1 and 3, and northwest of MBR 4. All conditions are satisfied, so the node is valid. The node in Figure 2(c) is invalid. In this case, MBR 4 is in location (1,1), which means that its centroid is southeast of the centroid for MBR 3. However, MBR 4 is northeast of MBR 3. Therefore, the node is invalid.

4 Search Strategy

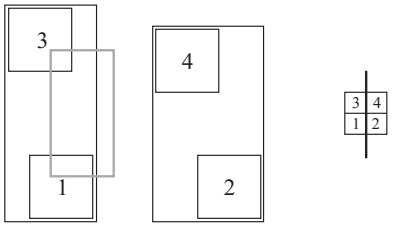
This section defines a binary search in two dimensions, then presents the 2DR-tree binary search strategy. The strategy works for region, point, containment, enclosure, and equality searches. We present the strategy for the region search.

The 2DR-tree binary search strategy extends the traditional one-dimensional binary search for arrays to work in a 2-dimensional node that stores MBRs. In a one-dimensional binary search, a search value is compared with the value at the midpoint in the array. The search continues with the values in the left-half of the array if the search value is less than the midpoint value, and in the right half if the search value is greater than the midpoint value. The 2DR-tree uses a similar strategy. A search region is compared for overlap with each half of a node region. The difference is in how each half is formed - the node region is partitioned through the x-axis (X-partition) or the y-axis (Y-partition). The term "binary search" will be used throughout the rest of this paper to indicate the 2-dimensional definition.

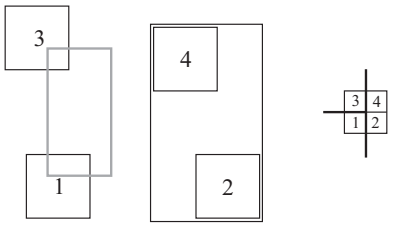
The search first performs a recursive binary partition of the node region in the root. The node region $(lx_{rt}, hx_{rt}, ly_{rt}, hy_{rt})$, which contains all locations in the root, is partitioned through the dimension with the longest range of index values. A partition through the x-axis at midpoint 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 at the midpoint cy_{rt} produces $(lx_{rt}, hx_{rt}, ly_{rt}, cy_{rt} - 1)$ and $(lx_{rt}, hx_{rt}, cy_{rt}, hy_{rt})$. If both dimensions have the same range of index values, then the x-axis is chosen by default. For each subregion, an MBR is created.



(a) Node and Data



(b) X-Partition



(c) Y-Partition

Figure 3. Recursive Binary Partitioning

Each MBR 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 MBRs, or the node region contains one location with an MBR. If one MBR is reached, it is tested for overlap with the search region. If overlap 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, it is tested for overlap with the search region and inclusion in the final result.

Figure 3 shows the recursive binary partition of a node. In Figure 3(a) the node region that contains all locations is $(0, 1, 0, 1)$. Figure 3(b) shows an X-partition of the node region and subsequent MBR creation. The query region, indicated with a light grey rectangle, overlaps the MBR corresponding to the first node subregion but does not

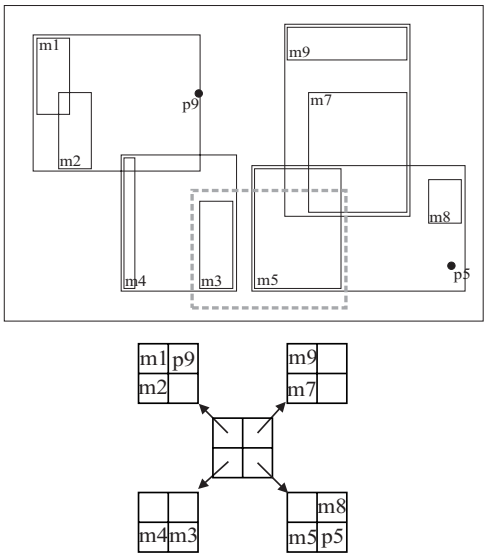


Figure 4. Region Query

overlap the MBR corresponding to the second node subregion. Figure 3(c) shows a Y-partition of this node subregion that leads to two more subregions, each with one MBR that overlap the query region. If the node is a non-leaf, the search continues in the corresponding subtree. If the node is a leaf, the objects are returned for the final result.

5 Example

This section demonstrates a 2DR-tree region binary search using the 2DR-tree in Figure 4. In the example, a subtree is displayed using the objects it contains, and one or more levels of parentheses to indicate its height. For example, (m4,m3) represents a leaf node (i.e.: height 0), while ((m4,m3),(m2,m1,p9)) represents a subtree of height 1, containing two MBRs that reference (m4,m3) and (m2,m1,p9), respectively.

The search first performs an X-partition of the node region $(0, 1, 0, 1)$ in the root. The X-partition is chosen because the node region has the same range of index values along the x-axis (X) and y-axis (Y). The X-partition 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)). MBRs are created for each subregion. Both overlap the query region, so partitioning continues first with $(0, 0, 0, 1)$. Since Y is now greater than X, a Y-partition is performed to produce $(0, 0, 0, 0)$ and $(0, 0, 1, 1)$. The subregion $(0, 0, 0, 0)$ contains one MBR which overlaps the query region. The search continues in the subtree (m4,m3).

An 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 MBR each, they both contain two locations, so the MBRs are created. The MBR for $(0, 0, 0, 1)$ does not overlap the query region, so it is not partitioned further. The MBR 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 MBR (m3) that overlaps the query region and is returned as part of the result. The subregion $(1, 1, 1, 1)$ contains no MBRs.

The search has finished with node (m4,m3) so it backtracks to subregion $(0, 0, 1, 1)$ in the root. It contains one MBR, ((m2,m1,p9)) that does not overlap the query region. The search moves to the MBR 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 an MBR 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 binary partition to locate m7 as the final qualifying object.

6 Evaluation

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

The performance evaluation uses several object sets to create the search trees. Each object set contains 100, 500, 1000, and 10000 squares respectively. Each square covers approximately 1% of the object space. All objects sets are uniformly distributed except one, which has an exponential distribution. Each set covers approximately 67-75% of the space with between 51-53% overlap. The binary search performance evaluation also uses several query sets, with each containing 1000 uniformly distributed rectangles or points. Each rectangle covers 5-10% of the space, where the extent of the space for each set matches the extent from a uniform data set. Although each set contains 1000 rectangles or points, only the first 100 are used in a test run. Having more objects in the query sets allows for future tests with more queries.

For each test run, 100 queries are executed on 1000 2DR-trees. Each tree is generated using a random sort of an 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 the number of seeks required to locate an object vs. the tree height. SAMs support retrieval of multiple objects and cannot guarantee a one-path search for region queries, point queries, and locating an individual object because many MBRs can overlap at a single point or region. Therefore the search may need to backtrack and check several paths to locate all qualifying objects. Ideally, the search locates an object and backtracks one level to locate another object. Pessimistically, the search must traverse several different paths - all of which originate at the root - to locate one object. 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.

6.2.1 Region Binary Search

Figure 5 and Table 1 show the results for the first set of test runs. Results in Figure 5 show that although the height of the 2DR-tree increases as the number of objects increases, the number of seeks required per object decreases. The more objects inserted, the better the 2DR-tree is able to cluster them so that fewer paths are searched to identify candidate objects. Therefore, the search is not required to backtrack all the way to the root to locate each object.

In Table 1, row 1 shows the average height for 1000 trees that each contains 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

#Objects	Height	#Seek/Sch	#Found/Sch	#Seek/#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

Table 1. Region Search with Varying Object Set Size

Dist'n	Height	#Seek/Sch	#Found/Sch	#Seek/#Found
Unif	12.31	40.94	6.90	5.93
Exp	12.31	29.11	8.28	3.52

Table 2. Region Search with Varying Distribution

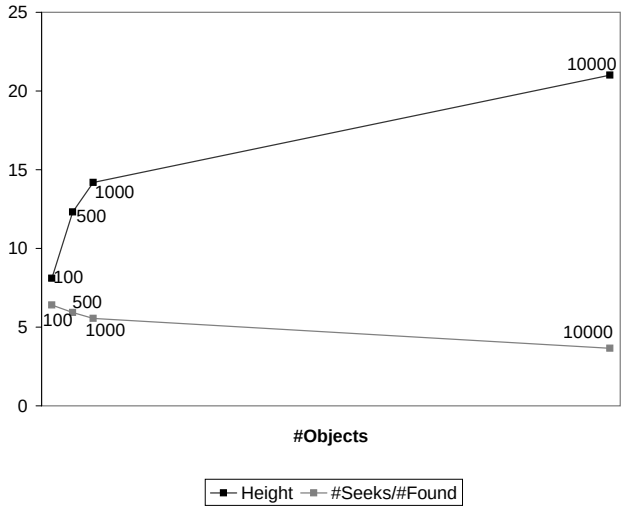


Figure 5. #Objects vs. Height and #Seeks/Object

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.

Table 2 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 the exponential distribution because the actual search results vary widely - many 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

Table 3 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.

Table 4 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 point is searching in less wasted space than in uniformly distributed data, and more

#Objects	Height	#Seek/Sch	#Found/Sch	#Seek/#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

Table 3. Point Search with Varying Object Set Size

Dist'n	Height	#Seek/Sch	#Found/Sch	#Seek/#Found
Unif	12.31	26.12	1.12	23.32
Exp	12.31	15.15	1.40	10.82

Table 4. Point Search with Varying Distribution

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

The 2DR-tree is a SAM that preserves spatial relationships between all objects by using nodes of the same dimension as the object space. We present the 2DR-tree binary search strategy. It supports region, point, containment, enclosure, and equality searches. Strategies for insertion, deletion and node validation, and a time complexity analysis, are available in [14, 15].

Experimental results show that the 2DR-tree is ideal for executing 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 zero area, 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 SAMs. The third is to develop algorithms for other types of spatial searches, such as adjacency, nearest neighbour, and spatial join. A final research direction is to improve support for both spatial networks [3] and spatial reasoning strategies for direction relations [4].

References

[1] Geological Survey of Canada, *Geoscience Data Repository*, http://gdr.nrcan.gc.ca/index_e.php, visited March 2006.

[2] V. Gaede and O. Günther, Multidimensional Access Methods, *ACM Computing Surveys*, 30(2), 1998, 170-231.

[3] S. Shekhar and S. Chawla, *Spatial Databases: A Tour* (New Jersey: Prentice Hall, 2003).

[4] D. Papadias, M. Egenhofer and J. Sharma, Hierarchical reasoning about direction relations, *Proceedings of the 4th ACM International Workshop on Advances in Geographical Information Systems*, Rockville, Maryland, USA, 1996, 105-112.

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

[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Atlantic City, USA, 1990, 322-331.

[7] T. Sellis, N. Roussopoulos and C. Faloutsos, The R+-tree: a dynamic index for multi-dimensional objects, *Proceedings of the 13th International Conference on Very Large Data Bases*, Brighton, England, 1987, 507-518.

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

[9] I. Kamel and C. Faloutsos, Hilbert R-tree: an improved R-tree using fractals, *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, 1994, 500-509.

[10] J. Orenstein, Redundancy in spatial databases, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, 1989, 294-305.

[11] R. Bayer, The universal B-tree for multidimensional indexing: general concepts, *Worldwide Computing and Its Applications, International Conference, WWCA '97*, Tsukuba, Japan, 1997, 198-209.

[12] N. Koudas, Indexing support for spatial joins, *Data and Knowledge Engineering*, 34(2), 2000, 99-124.

[13] D. Comer, The ubiquitous B-tree, *ACM Computing Surveys*, 11(2), 1979, 121-137.

[14] W. Osborn and K. Barker, The 2DR-tree: a 2-dimensional spatial access method, *University of Calgary Technical Report 2004-750-15*, 2004.

[15] W. Osborn, *The 2DR-tree: a 2-dimensional spatial access method*, (PhD Thesis, University of Calgary, 2005).