J. King Saud University, Vol. 21, *Comp. & Info. Sci.*, pp. 45-54,Riyadh (2009/1430H.)

A Theoretical and Empirical Evaluation of a Novel Spatial Data Indexing Structure

Ameur Touir
Department of Computer Sciences
College of Computer and Information Sciences
King Saud University
touir@ksu.edu.sa

(Received 20/01/2009; accepted for publication 18/03/2009)

Abstract. In this paper, we present a theoretical and empirical evaluation of the performance of the Multi Layer Quadtree (MLQ), a spatial join structure used for indexing spatial data. The Multi Layer Quadtree is based on the PM1-Quadtree. It permits the representation of multi-layers in a single structure rather than a structure for each layer. The theoretical evaluation of the MLQ is intended to compute the storage required by the structure in a worst case scenario to show that the structure would be accommodated in a reasonable memory without degrading the system performance. The empirical evaluation is based on real data as well as synthetic data to show the flexibility of the structure and the response attained by adopting such a structure. The application of the selection, insertion, and deletion operations using the MLQ is also presented. We consider point and range queries over two-dimensional data such as lines, poly-lines and regions data.

Keywords: Quadtrees, spatial join, multiway join, spatial access method, spatial query.

1. Introduction

Nowadays, the demand for Geographical Information Systems has increased which has brought about a variety of techniques for managing such systems. Among these worth noting techniques are spatial access methods. Spatial access methods are used to store, access and manage spatial data, such as points, lines, poly-lines, polygons, and surfaces. The data stored in spatial databases are considered as spatial objects. Querying spatial databases is complex and expensive. Queries include point-query, range-query and spatial join based query. Point queries involve the processing of a single dataset allowing the selection of a specific object. Range queries provide a set of objects that verify specific location-constraints. Examples of range queries are windowing and clipping. Range queries may be single-layered or multi-layered. Very frequently, in real life applications, we resort to use multi-layered queries in order to answer useful and frequent inquiries such as locating focal points (hotels, hospitals, etc.) with respect to other focal points. Such queries, referred to as spatial join queries, involve the joining of the implied layers. Other types of queries, such as topological queries including the nearest, neighbour of, or between may be based on single or multiple layers (Papadias et al., 2003). An example illustrating the implication of several layers is to request all schools located between two parks. The implied layers are the one containing the schools and the other one containing the parks.

When more than one layer is involved, the spatial join based query is triggered. It is a complex operation where several datasets are involved increasing thereby the complexity of the query.

Spatial join queries are commonly processed in several steps according to its spatial index structure. Among these spatial indexes, we note the R-tree and its variants R+-tree and R*-tree (Brinkhoff et al., 1993). R-trees use the minimum bounding rectangles (MBR) of the spatial objects to approximate them as a first filtering. A second filtering which requires a further processing is used on the geometry of the objects themselves. Another category of spatial indexes is the quadtree and its variants such as the PR quadtree and the PM quadtree (Samet and Webber, 1985). In contrast to the R-tree, Quadtrees operate on the geometry of the object itself and not on its approximation.

Further discussion of the spatial join operation is presented in section 2. Section 3 provides an overview of the Multi-Layer-Quadtree that were introduced in (Touir, 2004). Section 4 addresses the manipulation of such a Multi-Layer-Quadtree. Experimental results are presented in section 5. Section 6 concludes the paper.

2. Background and related work

A major problem in spatial join is dealing with the intermediary results in an optimized way. Several solutions were proposed. Patel et al. (1997) perform a pairwise spatial join algorithm using a multiprocessor environment to execute cascading joins. Jacox and Samet (2003) introduce the Iterative Spatial Join which is based on the sort-merge join. Their approach is best suited to cases where the intermediary results are already sorted. Papadopoulos et al. (1999) apply a two-join case study to evaluate the performance of four spatial join algorithms. Mamoulis and Papadias (1999) suggest a pairwise join method that combines pairwise join algorithm in processing trees. The leaves of the tree represent the relations indexed by R-trees and the intermediate nodes are join operators. Papadias et al. (1999) introduce another method of executing multilayer spatial join which consists of traversing synchronically all the spatial indices excluding combinations of intermediate nodes that do not satisfy the join condition. Mamoulis et al. (2004) study the complexity of the spatial queries that involve combinations of selections and joins. They focus on presenting a cost model that estimates the complexity of such queries; whereas in (Mamoulis and Papadias, 2002) the authors emphasize on the complexity of the joins that involve several layers. Park et al. (1999) highlights the different steps needed for such types of queries.

It is very common that the involved datasets are very large and so are the intermediary results. These results are not indexed in advance. Therefore, either a dynamic index is created (Papadopoulos et al., 1999) or no index. When no index is created, different ways take place to speed up the process. Patel et al. (1997) spatially sort the non-indexed objects by setting up leaves and match them to those of the existing tree that intersect it. Arge et al. (1998) propose a combination of plane sweep and space partitioning to join the datasets. Park et al. (2000) propose an improvement of the two-step processing in which the filter is separated from the refinement steps in the query optimization phase instead of the query execution phase.

A common issue faced while performing a multilayer spatial join is the utilization of as many datasets as the number of the involved layers. Each of these datasets is associated with a single tree used as a spatial. The organisation of spatial data objects requires the ability to cluster the objects according to their spatial location (Patel et al., 1997). Xiao et al. (2001) show how to cluster objects of different types to reduce the complexity of the operation. Touir (2004) introduces a novel approach of processing multilayer spatial join. It is based on the idea of using a single spatial index structure to perform the aforementioned queries. It is shown that the proposed index structure is capable of absorbing different datasets allowing for several types of queries to be performed. In this paper, we present a theoretical and empirical evaluation of the performance of the structure introduced in (Touir, 2004).

3. An Overview of the Multi Layer Quadtree (MLQ)

The MLQ is based on the concept of PM1-Quadtree (a variant of PM-Quadtree) (Samet and Webber, 1985). The MLQ is constructed successively from the PM1-quadtrees (t_1, …,t_n) that represent different layers (l_1, …, l_n) . For example, the MLQ in Fig. 2.a represents the dataset of layer l_1 and the MLQ in Figure 2.b is built upon the MLQ in Figure 2.a by inserting the dataset of layer l_2. Suppose that we have already inserted a layer (say l_1), and we would like to insert a new one (say l_2). The result of this second insertion will generate a virtual layer (say L) $L=l_1 \oplus l_2$. L is the superimposition of l_1 and l_2 as depicted in Fig. 1. The components of L are allocated to several levels. Figures 1, 2.a and 2.b show the construction of MLQ obtained by inserting two layers. Fig. 3 shows the resulting MLQ after the deletion of the object "line BC" of Fig. 1. More details are in (Touir, 2004).

The other aspect of the MLQ is its flexibility in handling queries. One of the important types of queries in this respects is the geometric selection (Rigaux et al., 2001; Sun et al., 2002) which can be point-based or window-based. These types of queries can target a single layer or several layers in which case the queries are considered as spatial join queries. Generally, the complexity of a query varies with the

number of the involved layers. However, in our case, all layers are in the same structure i.e. the MLQ.

Therefore, this complexity will not vary greatly when processing a single layer or multiple layers. All layers are implicitly involved in one structure. In region/window-based queries, the user may indicate the region where the search is going to be performed. Any object from any layer that intersects or is included in the specified region is considered as candidate object. The remaining conditions defined as predicates of the select criteria will then filter the desired objects among those candidate objects.

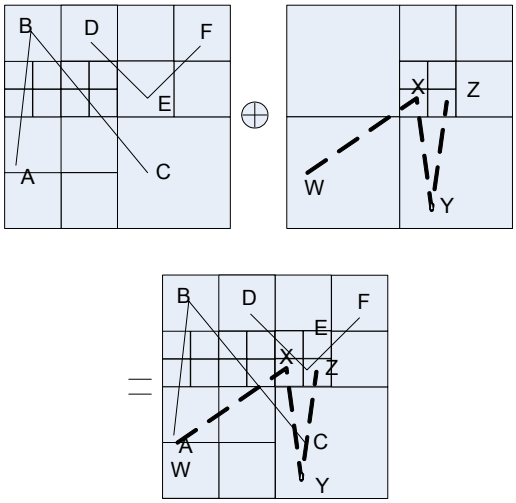
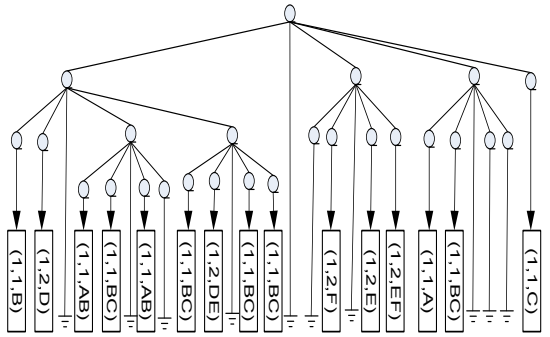
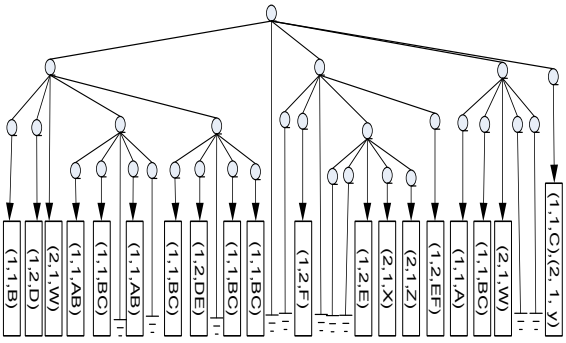


Fig. 1. Superimposition of two layers.



a. a. The resulting MLQ after inserting a first layer according to the example shown in Fig. 1.



b. The resulting MLQ after inserting a second layer according to the example shown in Fig. 1.

Fig.2. MLQ insert principle.

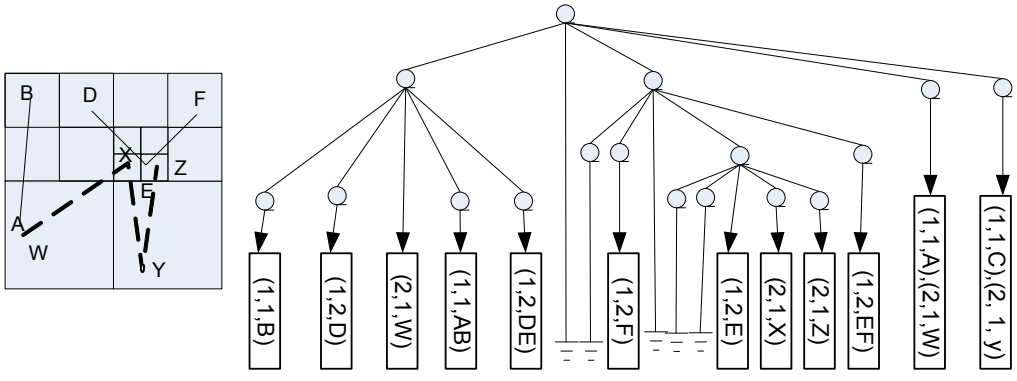


Fig. 3. The resulting MLQ after the deletion of an object (line BC of Fig. 1).

4. MLQ Storage Evaluation

Let us suppose that the processed layers are of size $2^N x 2^N$. N is the maximum number of subdivisions (levels) of any quadtree-like structure. When N= 10, layers are of size 1024x1024. Manipulating these layers (insertion, deletion, displaying, etc.) requires dealing with two structures: a front structure (MLQ) and a Rear structure (RS). Each node of the MLQ indexes a segment S of RS. If a segment S is full, a secondary segment is created and linked to S and so on.

Since the maximum number of nodes of a level i of a quaternary tree is 4^i, the maximum number of nodes of an MLQ of height N is $\sum_{i=0}^{N} 4^i = \frac{4^{N+1}-1}{3}$.

This implies that the RS has $\frac{4^{N+1}-1}{3}$ primary segments.

Contrary to the quadtree, where only three out of four sibling leaf nodes can contain data, the four sibling leaf nodes of a PM1Quadtree may contain data. Thus, the maximum number of leaf nodes of a PM1Quadtree is 4^N.

Let us consider that the maximum number of layers in the MLQ is L_{max}. Then, in the worst case, each of following the same reasoning as in (Vassilakopoulos and Manolopoulos, 1995), the number of segments needed to cover these components is:

$$NbSeg = \left\lfloor \frac{L_{Max}4^N - \left(\frac{4^{N+1}-1}{3}\right)}{S_{cap}} \right\rfloor .$$

We conclude that the total number of segments is bounded by:

$$TotNbSeg = \left(\frac{4^{N+1}-1}{3}\right) + NbSeg \qquad (1)$$

$$TotNbSeg \approx \frac{3L_{Max}4^N - 4^{N+1} + 4^{N+1}S_{cap}}{3S_{cap}}$$

$$TotNbSeg \approx \frac{4^N}{3S_{cap}}\left(3L_{Max} + 4S_{cap} - 4\right)$$

$$TotNbSeg \approx \frac{4^N}{3S_{cap}}\left(3L_{Max} + 4S_{cap}\right)$$

Therefore, $\lceil \log(TotNbSeg + 1) \rceil$ bits are sufficient to identify each segment. According to (1),

$$\lceil \log(TotNbSeg + 1) \rceil \approx \lceil \log(TotNbSeg) \rceil$$

$$\log(TotNbSeg) = 2N + \log(3L_{Max} + 4S_{cap}) - \log(3S_{Cap}) \quad (2)$$

For N=10, L_{max} =1024, and S_{cap}=16,

the number of bits needed by the front structure to address a component will be about 26 bits.

the L_{Max} layers is composed of 4^N components. This yields a $4^N L_{Max}$ components.

Let us associate one of these components to each of the MLQ nodes. In this case $4^N L_{Max} - \frac{4^{N+1}-1}{3}$ components are left with no segments.

To obtain the size of the segments' identifiers (pointers), we define S_{cap} as being the maximum number of components that can fit in a segment S of the rear structure RS.

Given that the total number of nodes is $\frac{4^{N+1}-1}{3}$, the MLQ size will be about 5MB which can fit easily into the main memory.

5. Experimental Results

In this section, we analyze the performance of the MLQ structure with a large set of real data and another set of synthetic (randomly generated) data. Data in this respect are considered as layers. Layers of different sizes were used to experiment and to study the behaviour of the structure. Fig. 6.a and Figure 6.c are respectively samples of real data, and synthetic data. The real data address five layers: Counties, states, rivers, drainage, and roads layers. They are taken from (esri).

The Synthetic data have been generated randomly by computing number of objects (polylines) that constitute a given layer along with their points. For each object, given the coordinates of an initial point P_0, P_i is randomly determined so that the distance between P_i and P_{i-1} is less than a given distance α in each direction (axis). The following

algorithms were used to generate the synthetic data.

```
void ComputeSyntheticData()
  Begin
        NbObjects = random(MaxObjects);
        i=0;
        while (i< NbObject) do
            NbPoints = random(MaxPoints);
            j = 0 ;
          Object_i.P_j = ComputeInitialPoint();
            while (j< NbPoints) do
            j= j+1;
                  Object_i.P_j=
        ComputeCoordinatesRandomly(Object_i.P_j-1);
            endwhile;
            i= i+1;
        endwhile;

  end

   Point ComputeCoordinatesRandomly(input Point
    P)
    begin
       P_1.x = random(P.x+β) ;
       while (abs(P_1.x-P.x)> α )
          P_1.x = random(P.x+β) ;
         P_1.y = random(P.y+β) ;
       while (abs(P_1.y-P.y)> α )
             P_1.y = random(P.y+β) ;
       return P_1;
  end;
```

For our experiment, one of the concerns is to study the growth of the MLQ after the insertion of each layer. For this purpose, we used several parameters as shown in table 1.

Table 1. Notations used in the experimental results

TotNbLines	The total number of lines that compose each inserted layer
$Node_{PMQ}$	The number of nodes that composes each PM1Q generated from the inserted layer
$Size_{PMQ}$	The size of each PM1Q in bytes
$Node_{MLQ}$	The total number of nodes of the MLQ, after the insertion of a new layer.
$Size_{MLQ}$	The size of the MLQ in bytes, after the insertion of each layer.
λPMQ	The number of non-empty PMQ leaves obtained from the inserted layer. This will illustrate the number of records in the rear structures that have to be updated with respect to each inserted layer
ΔMLQ	The increasing ratio of the MLQ after the insertion of each layer.

- The increasing ratio is defined as follows:

$$\Delta MLQ = \frac{Size_{MLQ_i} - Size_{MLQ_{i-1}}}{Size_{PMQ_i}},$$

where $Size_{MLQi}$ is the state (size) of the MLQ after inserting i layers, $Size_{PMQi}$ is the size of the PMQ of the i^{th} layer.
The fact that the number of the MLQ nodes is bounded implies that this ratio will ultimately be null after certain insertions. This will happen when MLQ reaches its maximum number of nodes:

$$Size_{MLQi} = \left(\frac{4^{N+1}-1}{3}\right).$$

Table 2 is divided into two parts. The first part highlights the behaviour of the MLQ structure with respect to the real data that were inserted. The second part highlights the behaviour of the MLQ structure with respect to the inserted synthetic layer.

The insertion of the first layer (counties) gives an increasing ration of 1(100%). This is due to the fact that the MLQ was empty and each inserted value generates a new node. The insertion of the second layer (states) gives an increasing ration of about 0.0013 (almost null), This means that only very few new nodes were generated, the other were only updated. This means that data of the second layer is almost distributed in the same location as the data of the first layer. The insertion of the fourth layer (rivers) generates an increasing ratio of about 0.2.This indicates that 20% of the inserted layer has generated new nodes in the MLQ, while 80% were stored in already existing nodes. This behavior is clearly observed with the synthetic data. Fig. 4.b and 4.d illustrated this observation. Also Fig. 4.a and 4.c show that the growth of the MLQ will ultimately reach an asymptotic limit regardless of the number of inserted layers.

Querying process:

As shown above, the MLQ size is bounded, whereas building an index for each layer generates as many indices as the number of layers. Fig. 4 illustrates the cumuli of the created PMQ. Generally, the number of nodes that composes a given index structure, influences the complexity of the queries.

Table 2. Experimental Results using real and synthetic data

	Experimental Result using Real Data (North America map)						
LayerName	TotNbLines	$Size_{MLQ}$	$Size_{PMQ}$	$Node_{MLQ}$	$Node_{PMQ}$	λPMQ	ΔMLQ
Counties	87151	2499312	1666208	52069	52069	39052	1
States	13851	2500656	647200	52097	20225	15169	0.001384
Roads	12654	2619696	683552	54577	21361	16021	0.116099
Rivers	6759	2794992	549664	58229	17177	12883	0.21261
Drainage	5415	2795376	431488	58237	13484	10113	0.000593
	Experimental Result using synthetic data						
LayerName	TotNbLines	$Size_{MLQ}$	$Size_{PMQ}$	$Node_{MLQ}$	$Node_{PMQ}$	λPMQ	ΔMLQ
Rand18	102327	5004150	5337760	166805	166805	122096	0.9375
Rand17	1456	5031390	142240	167681	4445	2077	0.191507
Rand16	98755	6522030	5300768	217401	165649	121928	0.281212
Rand15	2581	6596910	379456	219897	11858	5980	0.197335
Rand14	1848	6628230	221600	220941	6925	3293	0.141336
Rand13	16479	6697830	957568	223261	29924	17826	0.072684
Rand12	2292	6739350	256768	224645	8024	3758	0.161702
Rand11	20740	6909870	1695104	230329	52972	34017	0.100596
Rand10	2164	6936990	314240	231233	9820	4395	0.086303
Rand9	17236	7002990	985344	233433	30792	18575	0.066982
Rand8	3020	7052790	360192	235093	11256	5236	0.13826
Rand7	16867	7052790	961532	235093	31924	18015	0
Rand6	1471	7064670	155520	235489	4860	2207	0.076389
Rand5	76203	7379910	2957984	245997	92436	62799	0.106573
Rand4	3567	7431630	394272	247721	12320	5869	0.131178
Rand3	16762	7476030	989344	249201	30916	18380	0.044878
Rand2	800	7478310	119968	249277	3748	1737	0.019005
Rand1	22810	7478310	1128576	249277	35268	21432	0

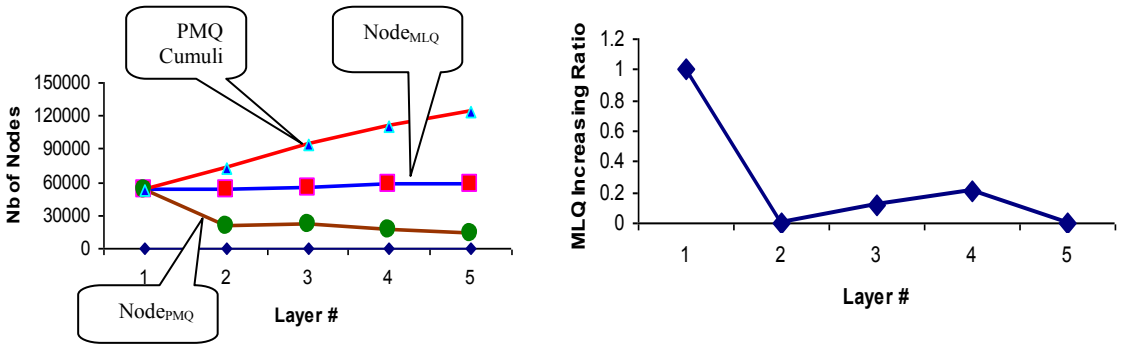
Table 2 shows example of the experimental result that we obtained from the layers that we dealt with in the MLQ. Two types of queries are performed and analyzed:

i. Point-based queries such as retrieve all objects of layer1 and layer3 that contain point (x,y): For this type of queries, several tests were randomly performed.
ii. Window-based queries such as retrieve all objects of all layers that intersect a given area. To analyze such type of queries, we performed the same window-queries using the MLQ generated from the test layers as shown in table 3. We compared the complexity of both structures (MLQ and PMQ) in term of I/O.

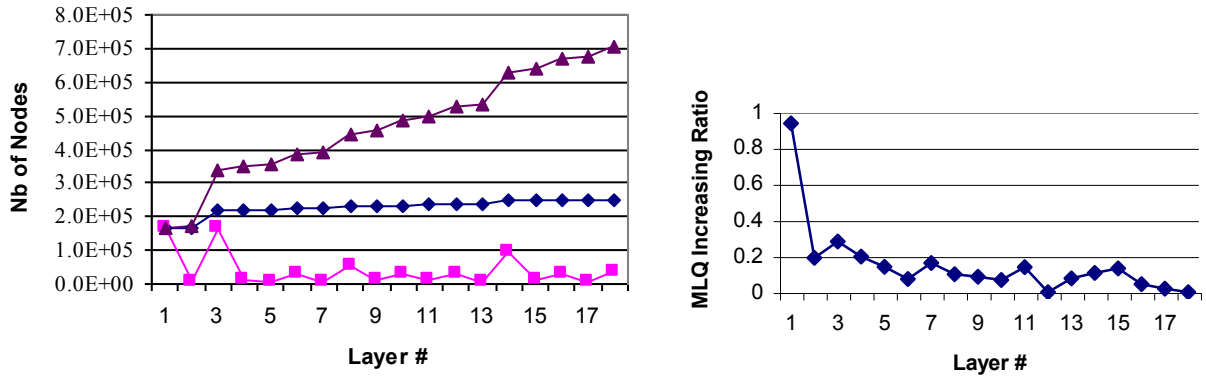
    Table 3 shows the number of disk accesses generated according to the size of the window specified in the query. Even though the MLQ aggregates layers into a single structure, we notice that its I/O performance is much more better than the PMQ when queries deal with multi-layers.
iii. Figure 5 is obtained by performing the same window-query on some PMQs and on MLQ. It illustrates the number of disk accesses needed to retrieve the requested data.

Table 3. I/O Performance of the MLQ and the use of independent PMQs

Length	Width	I/O(MLQ)	I/O(PMQs)
338	36	19671	28711
139	116	15083	22810
82	223	14966	22288
292	75	21158	31659
313	100	29764	50148
184	201	29688	46218
343	115	36915	55335
187	269	35571	48043



(a) (b)



(c) (d)

(a,c) Evolution of the MLQ in term of the number of nodes

(b,d) Behaviour of the Increasing ratio with respect to the inserted layers

Fig. 4. analysis of the evolution of the mlq with respect to the inserted layers shown in table 2 – c, d represent the experimental results using synthetic data.

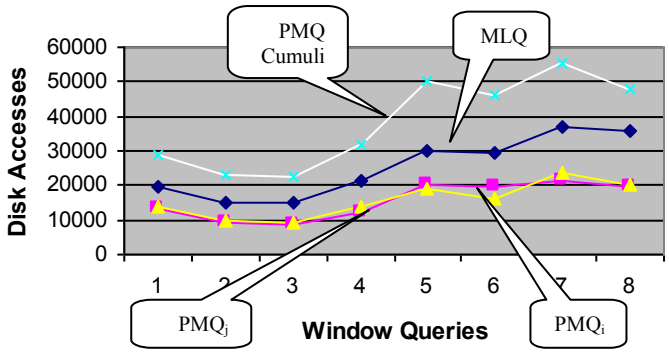
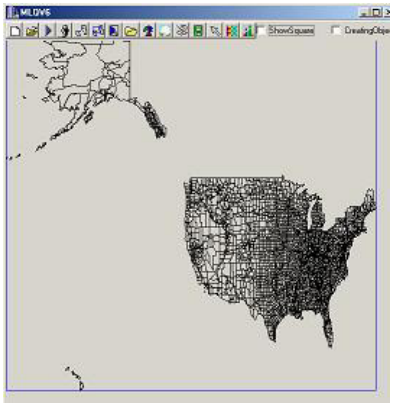
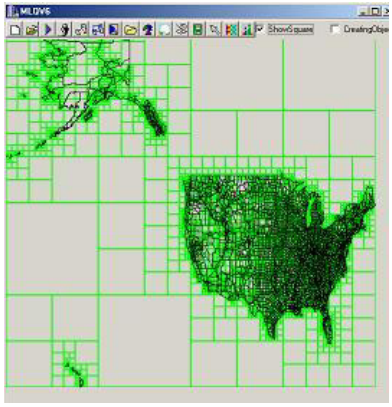


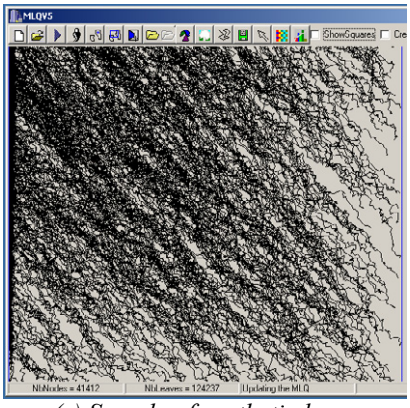
Fig. 5. Query Performance comparison of the MLQ



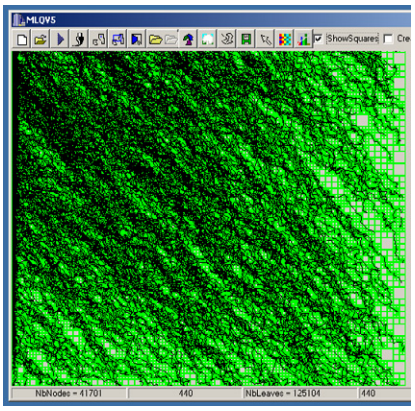
(a) Sample of the inserted layers (States, Counties, and Rivers)



(b) Illustration of Figure 6.a with the subdivision of the inserted layers



(c) Sample of synthetic layers



(d) Illustration of their subdivisions

Fig. 6. Some of sample layers on which the experimental results were conducted.

6. Conclusion

The paper presented a theoretical and empirical evaluation of the performance of the MLQ structure an overview of the multi-layer quadtree (MLQ) structure intended to facilitate spatial data operations. We highlighted the selection, insertion, and deletion operations for spatial data in view of the MLQ structure. We argue that the required main memory, that is required to accommodate the structure that indexes the data sets of several layers, is bound with the maximum number of nodes and the maximum number of layers. Considering that the maximum number of subdivisions (N) is10 and the maximum number of layers is 1024, we have shown that the required main memory is approximately 5MB. Such a memory requirement is reasonable in today's computing environment. Moreover, having a single indexing structure reduces the time needed to access data by reducing the traffic between the CPU and the disk storage.

We experimented with MLQ using real data and synthetic data. The empirical data confirmed a superior performance of the single indexing structure (the MLQ) compared to several indexing structures of the PMQ.

Currently, we are investigating the parallel processing of the MLQ. We intend to further analyze the behaviour of the MLQ in such a computing environment. We also intend to investigate its potential application in multi-version maps and/or map-history contents.

References

Arge, L.; Procopiuc, O.; Ramaswamy, S.; Suel, T. and Vitter, J.S. "*Scalable Sweeping-Based Spatial Join."* Proc. VLDB International Conference on Very Large Data Bases. New York City, USA, (1998), 570-581.

Brinkhoff, T.; Kriegel, H.P. and Seeger B. "*Efficient Processing of Spatial Joins Using R-trees*". ACM SIGMOD 22(2), (1993), 237-248.

Jacox, E. H. and Samet, H. "*Iterative spatial join*". ACM Transactions on Database Systems, 28(3), (2003), 230-256.

Mamoulis, N. and Papadias, D. "*Integration of Spatial Join Algorithms for Processing Multiple Inputs*". ACM SIGMOD 28(2), (1999), 1-12.

Mamoulis, N. and Papadias, D. "*Multiway Spatial Joins*". ACM Transactions on Database Systems, 26(4), (2002), 424-475.

Mamoulis, N.; Papadias, D. and Arkoumanis D. "*Complex Spatial Query Processing".* GeoInformatica 8(4), (2004), 311-346.

Park, H.; Lee, C-G.; Lee, Y-J. and Chung, C. "*Early Separation of Filter and Refinement Steps in Spatial Query Optimization*". Proc. International Conference on Database Systems for Advanced Applications. Kyoto, Japan, (1999), 161-168.

Park, H.; Lee, C-G.; Chung, C.W. and Lee Y-J. "*Spatial Query Optimization Utilizing Early Separated Filter and Refinement Strategy*". Information Sciences 25(1), (2000), 1-22.

Papadias, D.; Mamoulis, N. and Theodoridis, Y. "*Processing and Optimization of Multiway Spatial Joins Using R-trees.*" ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, Philadelphia, Pennsylvania, USA, (1999), 44-55.

Papadias, D.; Zhang, J.; Mamoulis, N. and Tao, Y. "*Query Processing in Spatial Network Database.*" Proc VLDB International Conference on Very Large Data Bases. Berlin, Germany, Morgan Kaufmann Publishers Inc., (2003), 802-813.

Papadopoulos, A.N.; Rigaux P. and Scholl, M. "*A Performance Evaluation of Spatial Join Processing Strategies.*" Proc. International Symposium on Advances in Spatial Databases, Hong Kong, China, (1999), 286-307.

Patel, J.; Yu, J.; Kabra, N.; Tufte, K.; Nag, B.; Burger, J.; Hall, N.; Ramasamy, K.; Lueder, R.; Ellmann, C.; Kupsch, J.; Guo, S.; Larson, J.; DeWitt, D. and Naughton J. "*Building a Scalable Geo-Spatial DBMS: Technology, Implementation, and Evaluation.*" Proc ACM SIGMOD Conference on Management of Data, Tucson, Arizona, USA, (1997), 336-347.

Rigaux, P.; Scholl, M. and Voisard, A. (ed.) "*Spatial Databases with Applications to GIS*", San Francisco, CA, USA Morgan Kaufmann, (2001).

Sun, C.; Agrawal, D. and El-Abbadi, A. "*Selectivity Estimation for Spatial with Geometric Selection.*" Proc. EDBT, International Conference on Extended Database Technology, Prague, Czech Republic, (2002), 359-360.

Samet, H. and Webber, R.E. "*Storing a collection of polygons using quadtrees.*" ACM Transactions on Graphics 4(3), (1985), 182-222.

Touir, A. "*ML-Quadtree: The Design of an Efficient Access Method for Spatial Database Systems."* King Saud Journal 17(10), (2004), 43-60.

Vassilakopoulos, M. and Manolopoulos, Y. "*Dynamic Inverted Quadtree: A Structure for Pictorial Databases."* Information Systems, 20(6), (1995), 483-500.

Xiao, J.; Zhang, Y. and Jia, X. "*Clustering Non-uniform-sized Spatial Objects to Reduce I/O Cost for Spatial-join Processing.*" The Computer Journal 44(5), (2001), 384-397.

www.esri.com

دراسة تقييمية و تجريبية لأداء الشجرة الرباعية الفضائية المتعددة الطبقات

عامر عبدالله طوير

قسم علوم الحاسب، كلية علوم الحاسب و المعلومات،

جامعة الملك سعود، الرياض،المملكة العربية السعودية

touir@ksu.edu.sa

(قدم للنشر في ٢٠٠٩/١٠/٢٠م؛ وقبل للنشر في ٢٠٠٩/٣/١٨م)

ملخص البحث. في هذه الورقة، نقدم تقييم نظري وتقييم عملي تجريبي لأداء إحدى تراكيب البيانات الفضائية والتي تسمى بالشجرة الرباعية متعددة الطبقات (MLQ). تستخدم هذه الشجرة الرباعية لفهرسة الخرائط و البيانات الفضائية والمكانية. كما ترتكز شجرة (MLQ) على شجرة معروفة باسم (PM1Q) . لدى شجرة (MLQ) ميزة معينة وهي أنها تستطيع فهرسة مجموعة من الخرائط في شجرة واحدة على عكس بقية التراكيب الفضائية المعروفة و التي تعتمد على تمثيل و ربط شجرة واحدة بكل خريطة. لقد تم دراسة أداء هذه الشجرة من الناحية الحجم والسعة المطلوبة في الذاكرة و الأداء و ذلك بتطبيقها على العديد من الخرائط الحقيقية و الافتراضية. كذلك تمت دراسة أدائها من ناحية تحديث البيانات (إضافة، حذف، تغيير) و كذلك من ناحية الاستعلامات وذلك من خلال البحث على أماكن معينة سواء أكان ذلك باستخدام البحث بالإطار أو بنقطة معينة.