Continuous Region Query Processing in the mqr-tree

Wendy Osborn
Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Alberta
T1K 3M4 Canada
wendy.osborn@uleth.ca

Abstract

This paper presents a novel approach for processing continuous spatial region queries using the mqr-tree. Previous approaches to this problem that utilize a spatial index have several limitations, including significant caching of data on the client, knowing the user trajectory in advance, sparse indices, and having to perform repeated searching whenever the location of the query region is updated. Taking advantage of desirable properties of the mqr-tree, a continuous region query strategy is proposed that does not require repeated searching from the top of the index, and also does not require the trajectory to be known in advance. I evaluate this strategy against repeated searching, and find that very significant savings are achieved with respect to the number of disk accesses required to fetch data. This results in a novel strategy that can be used for location-based services.

keywords: continuous region queries, location-based services, spatial access methods

1 Introduction

A location-based service provides results to a user of a mobile device (e.g. smartphone, tablet) based on their location, interests and the type of query being performed [11]. One example of such a query is a region query [10, 12]. A region query returns all points or objects that overlap a query region. With respect to location-based services, this can also be a region defined around a user and their current location. For example, a user may want to find all restaurants within a certain region around them. However, as they move around, the restaurants that would fall within this region will change.

Therefore, an efficient continuous region query strategy is important, especially when the query is initiated from a mobile device [11]. Many strategies have been proposed that can handle continuous region queries for location-based services. Several utilize spatial access methods [1], including [13, 14, 3, 5, 6, 9]. These strategies have several limitations, including repeating searching, caching more data on the device than is desired, or knowing the query trajectory in advance. The mqr-tree [7] is a spatial index with some desirable properties, including two-dimensional node and the maintenance of spatial relationships between objects. This structure lends itself to continuous spatial query process that achieves no repeated searching from the top of the index and no requirement to know the trajectory in advance.

In this paper, I propose a continuous query processing strategy that utilized the mqr-tree. The strategy is presented, followed by an evaluation of its server performance. It is found that the proposed continuous region query processing strategy significantly reduces the number of disk accesses (i.e. page hits) that are required to process a continuous region query, in both randomly and exponentially distributed data.

This paper proceeds as follows. Sections 2 and 3 summarize related work and required background. Section 4 presents and describes the continuous region query strategy. Section 5 presents the methodology and result of the performance evaluation. Finally, Section 6 conclude the paper and gives research directions.

2 Related Work

Many strategies have been proposed for processing a continuous region query (see Ilarri *et al.* [4]). Since my work resides in the area of strategies that utilize a spatial index [1], I focus my summary on these works. Some strategies that exist that utilize a spatial index, use an R-tree [2] or a modified grid file [8].

Song and Roussopoulos [13] propose an approach for continuous query processing that utilizes existing stationary spatial query approaches (e.g. [10]) to obtain a superset of m qualifying points, so that the result stays current while the query point moves around. An

978-1-943436-05-7 / copyright ISCA, SEDE 2016
September 26-28, 2016, Denver, Colorado, USA

issue with this approach is in choosing an appropriate value of m, so that fewer query calls are made but not at the expense of significantly increased storage at the client.

Tao *et al.* [14] propose a strategy that utilizes an R-tree for speeding up searches. One limitation of this approach is that "vertical" searching for new co-ordinates must be performed repeatedly. Lee et al. [6] improve upon this strategy by attempting to reduce the number of required queries by fetching both required and additional complementary objects. One issue is that repeated searching is still required to obtain enough complementary objects. An improvement was also proposed by Park et al. [9] by proposing a "horizontal" search for required objects along a trajectory. Although their strategy is shown to be efficient when spatial indexes are used, the entire trajectory must be known in advance.

A caching strategy is proposed by Hu et al. [3] which works for multiple spatial query types including region and nearest neighbour. Their strategy uses caching of previous query results and the R-tree nodes that lead to them. The cached R-tree nodes are always searched first, before fetching additional required objects from the server. One issue is that the caching overhead and local processing costs may be prohibitive.

Jung et al. [5] propose a continuous nearest neighbour approach using a grid index. Every cell in the index contains a minimum bounding rectangle (MBR), which encompasses the points that are reference by the cell. If a query region does not overlap the MBR, then none of its points will either. One limitation is that many portions of the index may be sparse.

One additional limitation of strategies that utilize a spatial index is that, whenever a continuous query result needs to be updated when the trajectory is not known in advance, a brand new search of the index needs to be initiated. The mqr-tree [7] is a spatial index that has desirable properties, which can be used to eliminate repeated searching but not require the trajectory to be known in advance.

3 The mqr-tree

In this section, I summarize some features of the mqr-tree that are required for this work. More details on mqr-tree validity, insertion, construction, and basic region searching algorithms can be found in [7].

The mqr-tree [7] is a spatial index that uses two-dimensional nodes to organize objects in two-dimensional space. This allows the existing spatial relationships between objects - and regions containing objects - to be maintained. After each insertion, a validity test ensures that all spatial relationships are maintained. This feature also allows the mqr-tree to support continuous region query processing.

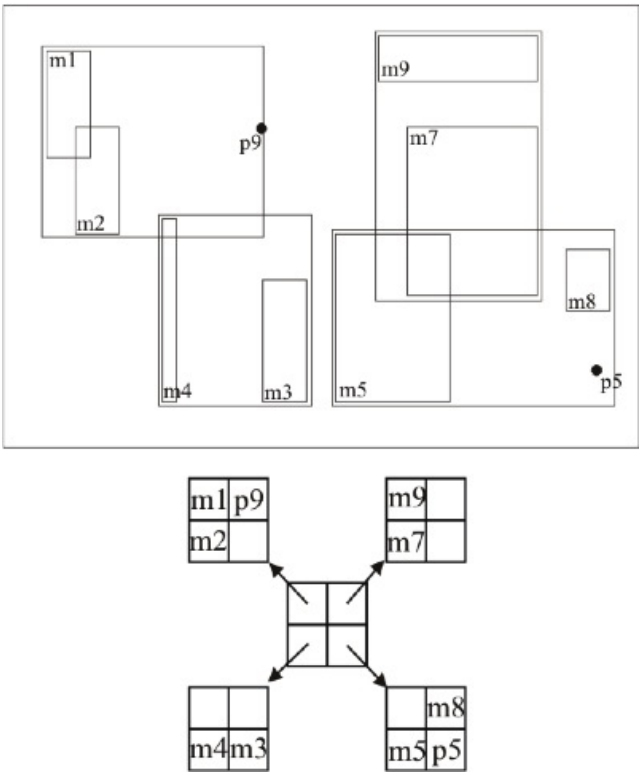

Figure 1: mqrtree Example (from [7])

Figure 1 depicts an mqr-tree structure for the given dataset. Every node has the quadrants NW, NE, SW and SE. There is also a CTR (i.e. centre) option, but is omitted for clarity and brevity. Associated with every node is a node MBR, which encompasses all objects and regions in the subtrees accessible from the node. The centroid of the node MBR aligns with the centre of the node. All references to objects and regions are placed in the appropriate quadrant based on their relationship to the centroid of the node MBR. For example, if we look at the leaf node containing m1,m2 and p9, we see that m1 is NW of the centroid for the node MBR that contains it, while m2 is SW and p9 is NE of the centroid respectively. Therefore, the objects are placed in the NW, SW, and NE quadrants, respectively, in the node. The other leaf nodes, and the root node, are organized in the same manner.

A region search is performed in the following manner. Beginning at the root node, if an object overlaps the query region, it is returned for the result. If a region overlaps the query region, then the search proceeds in the child node. This continues until all eligible paths have been searched and/or no other objects or regions

```
traversing = True
still_current = False

/*obtain first search region and initial starting point in mqrtree */
search_region = obtain_starting_region();
node X = search(Root, search_region);

While traversing = True
   /*is search within bounded region covered by current node?*/
   If search_region within node_MBR(X)

      /*if it is the same node_MBR as before, nothing to be done at this point,
        otherwise, find lowest descendent node_MBR that contains search_region */
      If still_current = False
          X = find_descendent_node_MBR(X)
      End If


   /*if search outside current region, move to parent region if exists.*/
  Else If has_parent_node(X)

     X = parent_node(X)
     still_current = false
     continue; /*this stops a new updated region query from being handled
                      until a new current node is found above*/

  /*otherwise, outside of space covered by index, so terminate search*/
  Else
     Exit
  End If

  //*obtain updated search region*/
  search_region = update_search_region()
End While
```

Figure 2: Continuous Region Query Strategy

overlap the query region.

In order to use the mqr-tree, it must be constructed. Constructing the mqr-tree may take anywhere from a few seconds for 10,000 objects, up to almost two minutes for 100,000 points, using repeated insertion of objects. However, the mqr-tree does support insertion and deletion operations, so changes to the data set can be accommodated quickly and easily.

4 Continuous Region Query Strategy

In this section, I present my continuous region query strategy, which takes advantage of the two dimensional nodes and spatial relationship management of the mqr-tree. The continuous region query strategy moves through the mqr-tree to update the query result on the client (e.g. mobile device), which results in huge savings with respect to the number of disk accesses (i.e. page hits) that are required for fetching results for the continuous query. I assume for this discussion that the strategy takes place on both the server and the client, although I am only evaluating the performance on the server for this work.

The strategy begins with a regular region search on the server, which identifies a node as a starting point for continuous navigation in the mqr-tree, along with the corresponding node MBR and a superset of objects that reside in the node MBR for the query result. The chosen node MBR serves as a super-region, where the moving query region and its result can be updated on

the client and be considered valid, as long as the query region remains inside of the super-region. This node MBR and superset are sent to the client.

When the location of the query region changes, if it is no longer fully contained within the super-region, then it is sent to the server so that an updated super-region and superset of objects can be found. However, instead of initiating a brand new query on the server, the update takes place by moving through the mqr-tree from the previous identified node, to a new node that contains the update query region. A new super-region and superset of objects are sent to the client. This process continues until the client stops the query, or the query region no longer resides in the overall space that is managed by the mqr-tree.

Figure 2 depicts the pseudocode for my strategy. At any given time on the server, the strategy is in one of several stages:

(1) The chosen tree node (and super-region and superset of objects) is still current on the client, and the client has not indicated that an update is necessary. Basically, nothing else happens at this point until the client indicates one is needed.

(2) The super-region is no longer valid, as indicated by the client. The client sends an updated query region to the server. The search for a new node (and super-region) begins at the last chosen node and proceeds by going up the tree to an ancestor node whose node MBR contains the query region. This may require only visiting the parent of the last chosen node, or it may require going up a few levels, and in the worst case going all the way to the root.

(3) A new initial node is found that contains the query region, but there may be a descendent node that fully encompasses the query region. The search contains by proceeding down the tree until the lowest descendent node is found that still contains the query region. At this point, the subtree from this node is traversed to obtain the superset of objects, and the super-region and superset are then sent to the client.

(4) The query region no longer fully resides in the overall space that is indexed by the mqr-tree. The search is terminated at this point.

It should be mentioned that, although it may be possible at any point that the search must proceed all the way back to the root, if the query region is corresponding to someone moving around - in particular, on foot - the location changes will likely be incremental and therefore it will likely not be necessary in most cases to have to proceed more than one or two levels up the tree.

5 Evaluation

In this section, I present the performance evaluation of the server portion of my continuous region query processing strategy. Using multiple query regions that reside on a trajectory, my strategy is compared to processing trajectories by repeated region query searching using the mqr-tree. I chose to compare my strategy in this way because the mqr-tree region query processing strategy was found to outperform that proposed for the R-tree [2], which is considered a benchmark spatial index and used by several strategies that are outlined in Section 2. I present the data and methodology used for the evaluation, followed by a discussion of the results.

The data used for this evaluation consist of both data and trajectories of query regions. Beginning with data sets, altogether, 40 data sets were used. They can be grouped into 4 categories, with each category ranging between 100 and 100,000 data items. The first category consists of randomly distributed squares of size 10x10 units. The second consists of randomly distributed points. The third consists of 10x10 squares that assume a exponential (i.e. skewed) distribution. Finally, the fourth consists of exponentially distributed points.

For each data set size (between 100 and 100,000), there is a corresponding trajectory of 10x10 squares, which will server as my continuous query regions. There is a total of 10 files, with each file containing $10 * \sqrt{datasetsize}$ squares.

Overall, 40 tests were carried out. For each test, an mqr-tree was constructed with a data set, then its corresponding trajectory was processed twice. For the first pass, the continuous region query processing strategy was applied to it. For the second pass, the trajectory was processed by performing a regular region query on each square. The data that was recorded for all tests was the total number of disk accesses (i.e page hits) required to process the trajectory. From this, I can also calculate the average number of disk accesses required per query region. I assume the worst case scenario that every time a node is accessed on the server, a disk access is required. However, no processing on the client adds to this cost.

I now present the results. Figures 3, 4 and 5 depict the total disk access results for the randomly distributed objects, randomly distributed points and exponentially distributed points. Figures 6, 7 and 8 depict the corresponding average disk access results. We can see in all cases that the continuous region query processing strategy for the mqr-tree significantly outperforms the processing of multiple query regions using repeated searching.

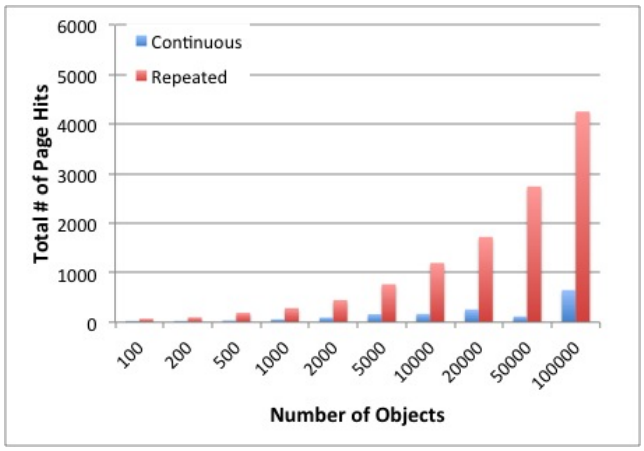

Figure 3: Randomly Distributed Objects

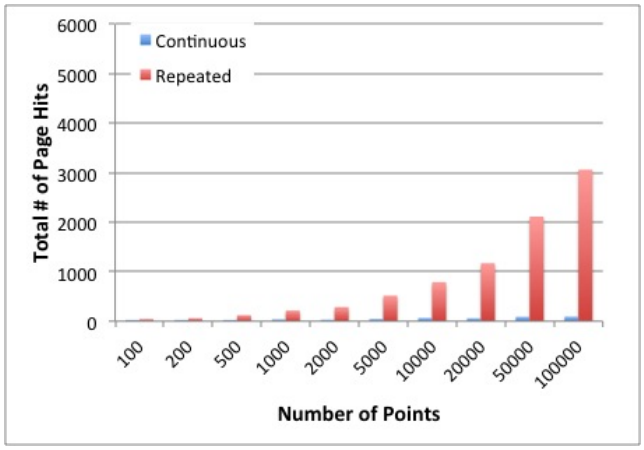

Figure 4: Randomly Distributed Point Data

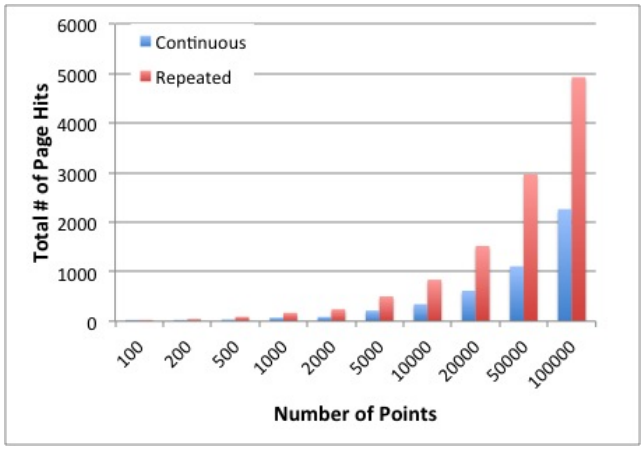

Figure 5: Exponentially Distributed Point Data

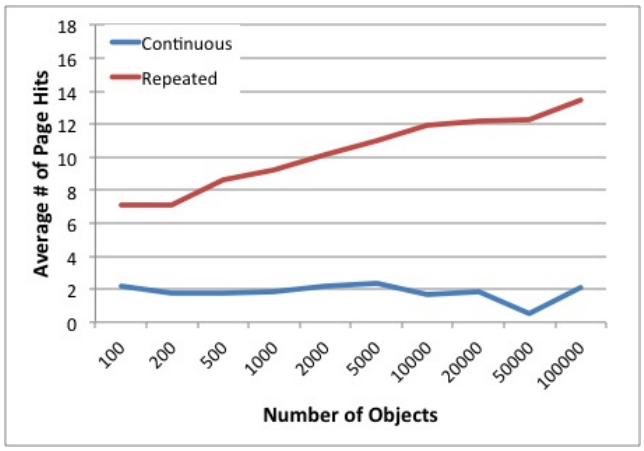

Figure 6: Randomly Distributed Objects

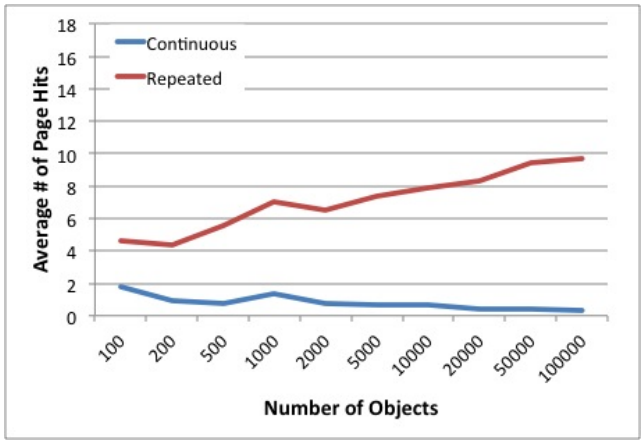

Figure 7: Randomly Distributed Point Data

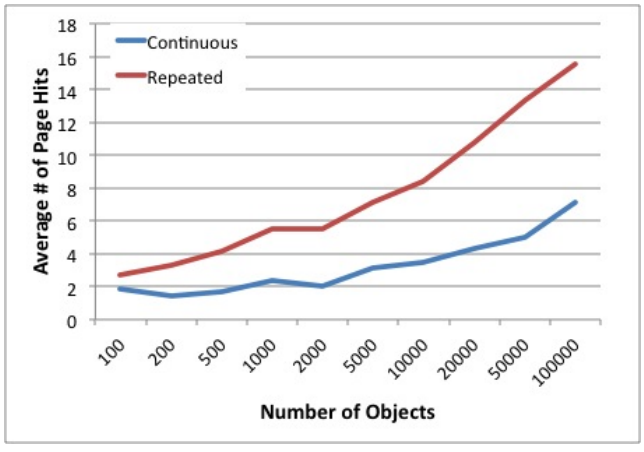

Figure 8: Exponentially Distributed Point Data

In particular, for the randomly distributed data sets, the savings are very large. Although the savings in the number of disk accesses are not as significant for the exponentially distributed point set, at least 50% savings is still achieved. With respect to the exponentially distributed square data sets, the savings achieved by the continuous strategy were so large that they did appear on the chart when placed next to results of the other approach, which is why they were left out of this paper.

6 Conclusion

The continuous region query strategy for the mqr-tree achieves the elimination of both repeated searching and the requirement of knowing the trajectory in advance. The performance evaluation show that, on the server side, significant savings are achieved in the number of disk accesses that are required to process a continuous region query.

Some future directions of research include the following. First, region queries that partially overlap (but are not fully contained in the space indexed by the mqr-tree) are not handled, and this must be addressed. Second, further evaluation versus other strategies (including ones that do not use an index) and real-life data sets is required. Third, evaluation of both the client side of our query processing strategy, and the entire system as a hole, is required. In particular, the execution time on the server, client and overall, must be evaluated.

References

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

[2] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pages 47–57, 1984.

[3] H. Hu, J. Xu, W.S. Wong, B. Zheng, D.L. Lee, and W.-C. Lee. Proactive caching for spatial queries in mobile environments. In *Proc. 21st Int't Conf. on Data Engineering*, 2005.

[4] S. Ilarri, E. Mena, and A. Illarramendi. Location-dependent query processing: Where we are and where we are heading. *ACM Comput. Surv.*, 42(3):12:1–12:73, March 2010.

[5] HR Jung, S.-W. Kang, MB Song, SJ Im, J Kim, and C.-S. Hwang. Towards real-time processing of monitoring continuous k-nearest neighbour queries. In *Proc. 2006 Int'l Conf. Frontiers of High Performance Computing and Networking*, 2006.

[6] K.C.K. Lee, W.-C. Lee, H.V. Leong, B. Unger, and B. Zhang. Efficient valid scope computation for location-dependent spatial queries in mobile and wireless environments. In *Proc. 3rd Int'l. Conf. Ubiquitous Information Management and Communication*, 2009.

[7] M. Moreau and W. Osborn. mqr-tree: a two-dimensional spatial access method. *Journal for Computer Science and Engineering*, 15, 2012.

[8] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, March 1984.

[9] Y. Park, K. Bok, and J. Yoo. An efficient path nearest neighbour query processing scheme for location-based services. In *Proc. 17th Int'l Conf. Database Systems for Advanced Applications*, 2012.

[10] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. *SIGMOD Rec.*, 24(2):71–79, May 1995.

[11] Jochen H. Schiller and Agnès Voisard, editors. *Location-Based Services*. Morgan Kaufmann, 2004.

[12] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2003.

[13] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *Proc. 7th Int'l Symp. on Advances in Spatial and Temporal Databases*, pages 79–96, 2001.

[14] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proc. 28th Int'l Conf. Very Large Data Bases*, pages 287–298, 2002.