

# Threshold Aggregate Query in Spatio-temporal Network Databases

Panfeng Zhou, Jun Gong

College of Computer and Information Science  
Northeastern University, Boston, MA, USA, 02115

{zhoupf,gjoliver}@ccs.neu.edu

***Abstract.** Aggregate R-B-tree (aRB-tree) provides a framework for supporting OLAP operations over spatio-temporal data. Although existing aRB-tree implementations process spatio-temporal aggregate queries pretty efficiently, it is hardly applicable for online processing due to the excessive accesses to the aggregate B-trees of the entries in the R-tree and the overlap in the R-tree. This paper addresses these problems by proposing several techniques that completely avoid the overlap in the R-tree and significantly reduce the excessive accesses to the B-trees during the update. We further extended original aRB-tree to aRBB<sup>+</sup>-tree which can process the threshold aggregate queries more efficiently.*

## 1. Introduction

Aggregate R-B-tree (aRB-tree) [6] provides a framework for supporting OLAP operations over spatio-temporal data. Efficient update is very important for real-time calculation in the OLAP operations. We address efficient update methods that will have less page accesses than the original methods proposed with aRB-tree.

Many research literatures assume that objects can move only along the fixed static spatial network (e.g., road network). We can optimize the index methods and query processing techniques to handle this important movement scenario. In the real-life applications, we often want to find the aggregate data with some threshold meeting certain spatio-temporal conditions. This special type of OLAP operations can be called threshold aggregate queries. We will use Crowded Roads Query (CRQ) as an example to illustrate the efficient processing of threshold aggregate query in spatio-temporal network databases. The input of the CRQ consists of a query spatial range  $S$ , a query time interval  $T$  and a density threshold  $\alpha$ . The output of the query contains the IDs of all the road segments whose densities had ever been bigger than  $\alpha$  during time interval  $T$  in the spatial range  $S$ .

In this paper, we assume that each road segment will report its density data with the form  $(t_i, r_j, \text{MBR}, \text{Data})$ , where  $t_i$  is the sample time instant,  $r_j$  is the road segment ID, MBR is the minimum bounding rectangle and Data is the road segment's density data.

The key contributions of this paper are as follows:

- 1) We introduce the RB-tree to avoid the overlap in the R-tree; we propose bulk-update and lazy update to reduce the accesses to the B-trees of index and leaf entries of the R-tree.

- 2) We introduce an important type of OLAP operations, threshold aggregate queries, which requires the aggregate result for some spatio-temporal conditions plus certain threshold.
- 3) We provide detail theoretical analysis to support our claims.

## 2. Related Work

This section first describes how to index the road network with an R-tree and how to index time-evolving density data with the aggregate B-tree. It then proposes how to use aRB-tree to solve the CRQ problems and limitations of original aRB-tree.

### 2.1. Road Network R-tree and Aggregate B-tree

Road network can be indexed with spatial index methods [2]. Given a road network, junction nodes, start/end nodes of a road segment and application dependent nodes (e.g. the turning points) can be identified. The road segment is defined as the road between two nodes. The road segment is not necessarily a line segment, it can be a poly-line or curve. We can use an R-tree [3, 1] to index the road segments due to its popularity in both commercial DBMS and academic researches. Specifically, the summary data of the sub B-trees are stored at the index entry of the R-tree as it is described in [5].

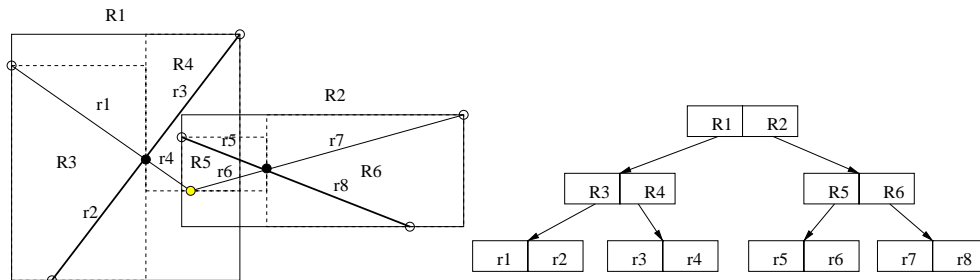


Figure 1: Index road network with R-tree

Figure 1 shows an example road network and its corresponding R-tree structure. Junction nodes, start/end nodes, and application dependent node are colored black, white, and gray respectively. The road network is divided into 8 road segments (r1 to r8). Each road segment r is enclosed by an MBR (Minimum Bounding Rectangle) and each road segment corresponds to one leaf node in the R-tree. Each road segment should be assigned a unique ID. We can order the road segments in Hilbert order according to the middle points of the road segments.

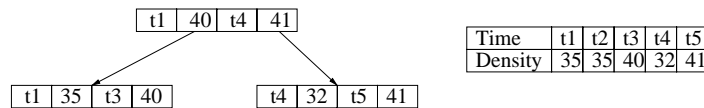


Figure 2: Max Density B-tree

In [8], Salzberg and Tsotras surveyed common temporal access methods in detail. For road segment r here, r's own density data needs to be stored; for spatial region R, the aggregate value of all the road segments inside R needs to be stored. As [7] suggested, if the value of density does not change in the consecutive sample time instants, no new leaf entry will be inserted into the B-tree. More details about processing the aggregations over general temporal data can be found in [4, 9]. Figure 2 illustrates the density change of one road segment r and r's max density B-tree.

## 2.2. aRB-tree and its limitation

The aRB-tree [6] provides a framework for supporting OLAP operations over spatio-temporal data. It indexes the spatial regions with an R-tree. Each entry of the R-tree (both leaf entry and index entry), has a pointer to an aggregate B-tree. R-tree's entry  $r$  has the form  $\langle r.MBR, r.pointer, r.BTree, r.aggr[] \rangle$ , where  $MBR$  and  $pointer$  have the same meaning as they are in the normal R-tree [3, 1];  $r.aggr[]$  keeps the aggregate data of  $r$  over all time stamps;  $r.BTree$  is a pointer to the aggregate B-tree which keeps historical data of  $r$ . Each aggregate B-tree entry  $b$  has the form  $\langle b.time, b.pointer, b.aggr[] \rangle$ , where  $b.aggr[]$  is the aggregate data for  $b.time$  as it is illustrated in figure 2.

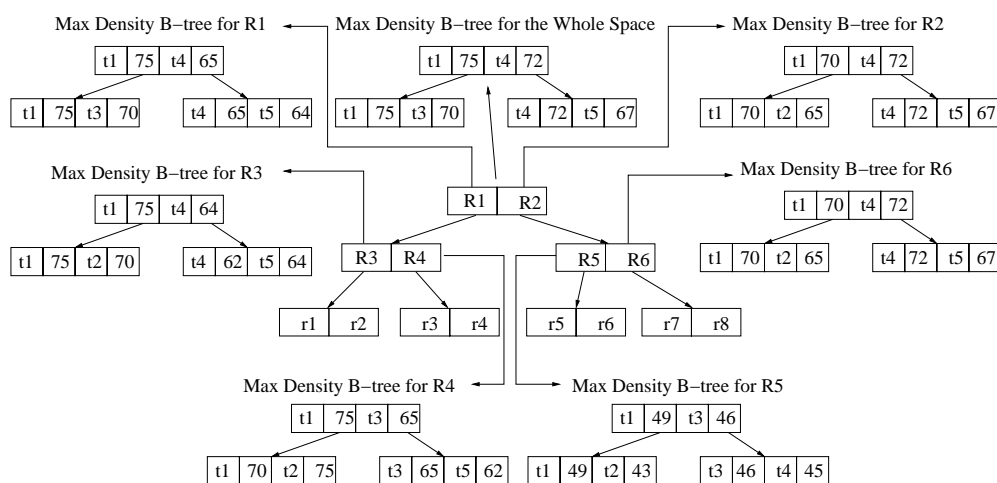


Figure 3: Max Density aRB-tree

In Figure 3, we show an example aRB-tree, Because of the space limitation, the max density B-trees of the leaf entries are not shown in the figure.

Let us now consider that one CRQ is looking for all the road segments (IDs), who intersect with the query spatial range  $S$  (dashed), and whose densities during the time interval  $[t1, t3](T)$  are bigger than the density threshold  $50(\alpha)$ , as illustrated in Figure 4.

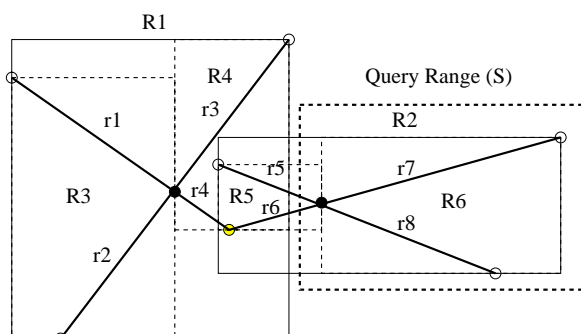


Figure 4: CRQ spatial range

Search starts from the root of the R tree. Entry R1 is outside the query spatial range  $S$ , so its subtree can be pruned. Entry R2 partially overlaps with the  $S$ , so its max density B-tree is retrieved. We find R2's max density during  $T$  is 70 which is bigger than  $\alpha$ , so we need to access R2's subtree. Similarly, R5's subtree can be pruned, R6's subtree and  $r7, r8$  are retrieved. The final result is  $r7, r8$ . The query algorithm can be summarized as only visiting  $r$ 's subtree when  $r$ 's MBR intersects with  $S$  and  $r$ 's max

density during  $T$  is bigger than  $\alpha$ . We will address an alternative method which can prune more branches in the later sections.

The aRB-tree update includes the update of both R-tree and B-trees. If there is not any overlap in the R-tree, we only need three page accesses to locate  $r_5$ 's position in the R-tree. But due to the overlap, we may only update R-tree's index entries while tracing back, thus many extra page accesses are needed. We introduce a new RB-tree to completely avoid such overlap problem for the update of the static road network R-tree. On the other hand, the update of all the density data requires another excessive accesses to the B-trees of R-tree's index entries. We will use *bulk update* to solve this problem. Finally, in real-life applications, the density data will not change all the time. We will use *lazy update* to take advantage of this characteristic.

### 3. aRBB<sup>+</sup>-tree

#### 3.1. RB-tree

As we already mentioned, each road segment should be assigned a unique ID. If we assign random IDs to the road segments before we create the R-tree, the IDs of the road segments in the same leaf page will be discrete, so when new density data arrives, due to the overlap in the R-tree, we can not decide which subtree the upcoming road segment belongs to. In order to solve this problems, we use a simple, but effective method: label the road segments with consecutive integers after we create the R-tree and store the ID range in the index entry. Each index entry  $r$  has the form  $\langle r.MBR, r.pointer, r.BTree, r.aggr[], r.startID, r.endID \rangle$ , where *startID* and *endID* record the smallest and biggest road segment ID in  $r$ 's subtree.

When a new density data arrives, we can compare its road segment ID with the ID range of each index entry and decide which subtree to go down. This will completely avoid visiting multiple pathes in the R-tree since the ID range in each entry do not have any overlap and we only need one travel from root to leaf to update the corresponding B-trees. Since this method combines the R-tree and B-tree in the same tree structure, we name it as RB-tree

#### 3.2. Bulk Update

The density B-tree of the index entry  $r$  will be accessed multiple times at each time instant since  $r$  needs to keep the update density information of its subtrees. We hereby introduce three new fields to the index entry to solve this problem. These three fields are *TotalCount*, *CurrentCount* and *CurrentMax*. For each index entry  $r$ , *TotalCount* records the number of road segments in  $r$ 's subtree. *CurrentCount* records the number of road segments in  $r$ 's subtree that have been updated for the current time instant. *CurrentMax* records the maximum density for current time instant. *TotalCount* will not change during the lifetime of an R-tree. *CurrentCount* is initially set to 0 and be increased by 1 each time a road segment in  $r$ 's subtree get updated. *CurrentCount* will be reset to 0 when it equals to *TotalCount*. *CurrentMax* keeps track of the biggest density data seen so far. Therefore the maximum density data only needs to be updated once after all the data have arrived instead of being updated each time when a new data arrives before.

### 3.3. Lazy Update

Bluk Update aims at reducing the accesses to the B-trees of the index entries in the R-tree. This section introduces Lazy Update which will benefit the B-tree accesses of both index entries and leaf entries in the R-tree. There are some unnecessary page accesses for the B-trees if the density does not change in the consecutive time instant. In order to solve this problem, we will change the usage of the field ,  $r.aggr[]$  in the original aRB-tree. The  $r.aggr[]$  in the original aRB-tree stores the summarized data of  $r$  accumulated over all the time stamps. But the aggregate value over all the time stamps is not so useful in practice since we rarely ask for the highest density over the whole time interval. We will use  $r.aggr[]$  to store the most recent density value which has been inserted into  $r$ 's B-tree. For each index entry  $r$ , when  $TotalCount = CurrentCount$ , we will compare  $CurrentMax$  with  $r.aggr[]$ . If they are equal, we do not need to update  $r$ 's B-tree. We only update  $r$ 's B-tree when the max density changed since the last time instant. For each leaf entry, we do similar change.

So far, we optimize the update of aRB-tree in three ways: use RB-tree to avoid the overlap in the R-tree which in turn reduces the cost of page access for R-tree leaf nodes; use Bulk Update to reduce the accesses to the B-trees of the index entries in the R-tree; use Lazy Update to reduce the accesses to the B-trees of both leaf and index entries.

### 3.4. Efficient Query

In this section, we will illustrate how to further extend the original aRB-tree to make the query processing of threshold aggregate query even more efficiently.

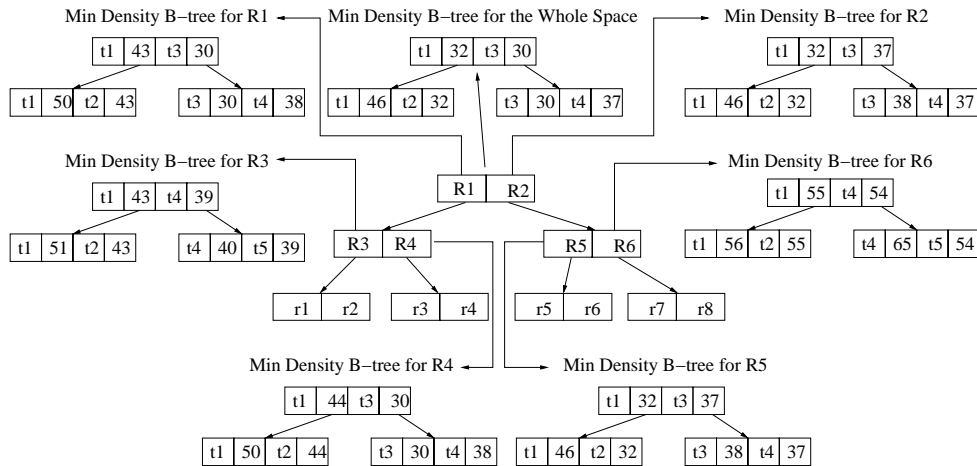


Figure 5: Min Density aRB-tree

We observe that an index entry  $r$ 's subtree doesn't need to be visited if the minimum density of  $r$  during the query time interval  $T$  is bigger than  $\alpha$  and  $r$  is completely inside the query spatial range  $S$ , since all the road segments in  $r$ 's subtree meets the query requirements. Intuitively we can store the minimum density to speed up such query. The improved query algorithm can be described as follows: start from the root, check each entry  $r$ , if  $r$ 's MBR does not intersect with  $S$ , prune  $r$ 's subtree; if  $r$ 's MBR intersects with  $S$ , but its max density during  $T$  is smaller than  $\alpha$ , prune  $r$ 's subtree; if  $r$ 's MBR intersects with  $S$ , its max density during  $T$  is bigger than  $\alpha$ , and its minimum

density during  $T$  is also bigger than  $\alpha$ , then add all the road segments in  $r$ 's subtree to the result set and prune  $r$ 's subtree.

In the implementation, we choose to store the maximum and minimum values in one single max/min B-tree instead of saving them in two separated B-trees for the following two reasons: 1. Store max and min values in one single B-tree requires less B-tree accesses during the update. 2. store maximum and minimum together will let them share some data such as time stamps, which will save some storage space.

#### 4. Conclusions

In this paper, we address the threshold aggregate query in spatio-temporal network databases. The aRB-tree is extended to aRBB<sup>+</sup>-tree which can efficiently process the threshold aggregate query. We propose three methods for efficient update of aRBB<sup>+</sup>-tree. The RB-tree is used to eliminate the overlap in the R-tree. The bulk update is used to reduce the accesses to the B-trees of the index entries in the R-tree. The lazy update is used to reduce the accesses to the B-trees of both index entries and leaf entries in the R-tree. We also propose a query method to efficiently query the aRBB<sup>+</sup>-tree.

#### References

- Beckmann, N., Kriegel, H., Schneider, R. and Seeger, B. (1990) "The R\*-tree: An Efficient and Robust Access Method for Points and Rectangles", in Proceedings of ACM/SIGMOD Annual Conference on Management of, 322–331.
- Gaede, V. and Gunther, O. (1998) "Multidimensional Access Methods", in ACM Comput. Surv., 30(2), 170–231.
- Guttman, A. (1984) "R-trees: a dynamic index structure for spatial searching", in Proceedings of ACM/SIGMOD Annual Conference on Management of Data, 47–57.
- Kline, N. and Snodgrass, R.T. (1995) "Computing Temporal Aggregates", in Proceedings of International Conference on Data Engineering, pp 222–231.
- Papadias, D., Kalnis, P., Zhang, J. and Tao, Y. (2001) "Efficient OLAP Operations in Spatial Data Warehouses", in Proceedings of Symposium on Spatial and Temporal Databases, pp 443–459.
- Papadias, D., Tao, Y., Kalnis, P. and Zhang, J. (2002) "Indexing Spation-temporal Data Warehouses", in Proceedings of International Conference on Data Engineering, pp 166–175.
- Salzberg, B., Jiang, L., Lomet, D., Barrena, M., Shan, J. and Kanoulas, E. (2004) "A Framework for Access Methods for Versioned Data", in Proceedings of International Conference on Extending Database Technology.
- Salzberg, B. and Tsotras, V.J. (1999) "Comparison of Access Methods for Time-Evolving Data", in ACM Comput. Surv., 31(2), pp 158 – 221.
- Yang, J. and Widom, J. (2001) "Incremental Computation and Maintenance of Temporal Aggregates", in Proceedings of International Conference on Data Engineering, pp 51–60.