Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 49 (2007) 324–331
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Indexing range sum queries in spatio-temporal databases

Hyung-Ju Cho, Chin-Wan Chung *

Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, 373–1 Kusong-dong, Yusong-gu, Taejon 305–701, Republic of Korea

Received 4 August 2005; received in revised form 8 May 2006; accepted 25 May 2006
Available online 10 July 2006

Abstract

Although spatio-temporal databases have received considerable attention recently, there has been little work on processing range sum queries on the historical records of moving objects despite their importance. Since the direct access to a huge amount of data to answer range sum queries incurs prohibitive computation cost, materialization techniques based on existing index structures are suggested. A simple but effective solution is to apply the materialization technique to the MVR-tree known as the most efficient structure for window queries with spatio-temporal conditions. Aggregate structures based on other index structures such as the HR-tree and the 3DR-tree do not provide satisfactory query performance. In this paper, we propose a new index structure called the Adaptively Partitioned Aggregate R-Tree (APART) and query processing algorithms to efficiently process range sum queries in many situations. Our experimental results show that the performance of the APART is typically 1.3 times better than that of its competitor for a wide range of scenarios.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Spatio-temporal database; R-tree; Range sum query; Indexing technique

1. Introduction

With the rapid increase of applications that deal with moving objects (e.g., intelligent traffic systems and mobile communication systems), the spatio-temporal aggregation has become one of the predominant operators for data analyses. Specifically, spatio-temporal databases (STDB) record a history of the behaviors of moving objects. In addition to traditional moving objects such as planes, people, and cars, the events that occur in time space can also be regarded as moving objects since they can be captured and identified by the combination of spatial and temporal information. Examples include mobile phone calls and traffic accident records.

Although several researchers addressed indexing and processing of timestamp queries and time interval queries [8,12,14,17], navigational queries [11], and nearest neighbor queries [6,13] in STDB, there has been little work on range sum queries for moving objects [15]. Similar to multi-dimensional databases such as OLAP (On-Line Analytical Processing) systems, range sum queries are also crucial in many real-life spatio-temporal applications. Let us present two real-life examples of range sum queries as follows:

- Q1: What is the total number of outgoing phone calls made from within Central Park during the previous month?
- Q2: What is the total number of traffic accidents within a 1 km radius of each school in June 2005?

Such information may be especially useful to estimate future states and to generate good future plans. For example, Q1 allows mobile communication companies to avoid potential network congestion and determine locations for new base stations. Such kinds of data include mobile phone calls and traffic accident records which are significant enough to be stored in databases.

* Corresponding author.
E-mail addresses: hjcho@islab.kaist.ac.kr (H.-J. Cho), chungcw@islab.kaist.ac.kr (C.-W. Chung).

0950-5849/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2006.05.005

Although range sum queries can be processed simply using operational databases, their computation is expensive due to the direct access of potentially huge amounts of data, rendering online processing inapplicable. To efficiently process these queries, materialization techniques were suggested. However, the data cube [3,5] employed by traditional OLAP systems is not suitable for range sum queries in STDB due to the lack of predefined containment hierarchies (e.g., day/week/month) [7,10]. Thus, for the case of range sum queries in spatial and spatio-temporal applications, the materialization techniques (i.e., pre-aggregation techniques) based on index hierarchies are preferable [7,10]. For the latter, it is reasonable to consider the idea of materialization to the MVR-tree [14] which is known as the most efficient index structure for the historical data. Unfortunately, it cannot provide satisfactory performance on range sum queries with a long time interval, which are frequently required. The reason is that multiple R-trees should be involved to answer them. To overcome this, the aRB-tree and its modifications (i.e., the aMVRB-tree and the a3DRB-tree) are proposed in [15]. The intuition behind them is that, by keeping additional summarized information inside the index, aggregation queries with arbitrary groupings can be answered by the intermediate nodes, thus saving accesses to detailed data.

Unlike spatial data, spatio-temporal data consist of spatial and temporal information. Special attention should be paid to the difference between the spatial dimension and the time dimension. Since the historical behaviors of moving objects are recorded chronologically within a fixed spatial area, the time region to be indexed increases as time passes while the spatial region does not. Thus, the time region should be partitioned sophisticatedly and the partitioned regions should be indexed by multiple R-trees.

This work is based on our previous work [2] which presented an adaptive indexing technique using query workloads for timestamp queries and time interval queries. In this work, we extend the adaptive partitioning method to range sum queries. To efficiently process range sum queries in a variety of scenarios, we propose an adaptive index structure, the *Adaptively Partitioned Aggregate R-Tree* (APART), which divides the time region dynamically using the query workload.

The techniques we will present in the paper can be applied to other aggregate functions such as *count* and *average*. *Count* is a special case of *sum* and *average* can be obtained by keeping the two columns (*sum*, *count*).

The rest of the paper is organized as follows: Section 2 briefly describes the existing spatio-temporal index structures and considers possible aggregate index structures for range sum queries in STDB. Section 3 presents the query-adaptive partitioning method, a range sum query processing algorithm, and construction of an auxiliary aggregate structure. Section 4 offers extensive experimental results for various query workloads and datasets. Finally, Section 5 provides conclusions.

2. Related work

For historical records of moving objects, several access methods have been developed. These include the 3DR-tree [17], the HR-tree [8], the MVR-tree [14], and the APR-tree [2].

Time interval queries refer to *window* queries with continuous timestamps while timestamp queries are *window* queries at a single timestamp. The 3DR-tree is a simple extension of the R-tree [1,4] in order to index spatial and time information simultaneously. It treats time as another dimension. Whenever an object moves to another position or changes its shape, a new MBR is created to represent the change of the object and the MBR containing both its spatial extent and lifespan is inserted into the 3DR-tree. The 3DR-tree is effective for time interval queries, but very poor for timestamp queries due to a large search space.

The Historical R-tree (HR-tree) [8] creates an R-tree whenever objects in the previous R-tree change their positions or shapes, but common branches of consecutive R-trees are stored only once in order to save space. The timestamp query is directed to the corresponding R-tree and the search is performed inside this tree only. Thus the timestamp query becomes the ordinary window query and is handled very efficiently. The time interval query should search the corresponding R-trees of all the timestamps involved.

The Multi-Version 3DR-tree (MV3R-tree) [14] is the access structure that combines the Multi-Version R-tree (MVR-tree) and a small auxiliary 3DR-tree built on the leaf nodes of the MVR-tree. The former is used to answer timestamp and short interval queries and the latter to answer long interval queries. Although the size of the auxiliary 3DR-tree is very small since it shares the leaf nodes of the MVR-tree, it improves the performance on interval queries. Compared with the HR-tree and the 3DR-tree, the MV3R-tree shows competitive performance on both timestamp and time interval queries by using multiple logical tree structures, each responsible for a fixed time interval.

The APR-tree [2] adaptively partitions the time domain using query workloads which include timestamp queries and time interval queries. Since the time domain of the APR-tree is automatically fitted to query workloads, the APR-tree outperforms the other access methods for various query workloads. Unlike the other access methods, the size and the update cost of the APR-tree are affected by query workloads. The update cost of the APR-tree is on the average similar to that of the 3DR-tree which has the smallest update cost. We extend the idea of the APR-tree to the APART for processing range sum queries. Unfortunately, in numerous OLAP applications where range sum queries are frequently employed, the average query interval length may not represent the actual query workload. For example, half of queries are very long interval and half of the queries are very short interval. In this case, the APART is not good for either kinds of the queries. To deal with this problem effectively, an auxiliary aggregate structure is introduced in the APART.

The R-tree is known to be one of the most popular index structures to efficiently process *window* queries in spatial databases. Intuitively, the aggregate R-tree (aR-tree) [7,10] improves the R-tree's performance in range sum queries by storing, in each intermediate entry, pre-aggregated sums of the objects in the subtree. Fig. 1 shows an example of an aR-tree. Note that in figures of this paper, numeric values in parentheses denote pre-aggregated sums. Given a range sum query, represented by the bold rectangle in Fig. 1(a), that corresponds to "find the sum of records which intersect q_R", we do not need to visit node N_2 even if this node is inside q_R. The reason is that $r_5 = 15$ already has the pre-aggregated sum of records covered by N_2. However, node N_3 should be searched since it is partially covered by q_R. Consequently, the query answer is 22.

Similar to the aR-tree, it is quite natural to transform existing spatio-temporal index structures to aggregate structures for range sum queries in STDB. The HR-tree and the 3DR-tree can adopt the pre-aggregation technique employed in the aR-tree without any difficulty. However, since the aggregate 3DR-tree (a3DR-tree) and the aggregate HR-tree (aHR-tree) are generalizations of the 3DR-tree and the HR-tree, respectively, they do not yield competent query performance.

In [15], Tao and Papadias proposed a unified solution which is currently regarded as the best approach for historical spatio-temporal aggregation. Specifically, the proposed solution consists of two types of indexes: (i) a host index which is an aggregate spatial or spatio-temporal structure managing region extents and (ii) numerous measure indexes (one for each entry of the host index) which are aggregate temporal structures storing the values of measures during the history. Variations of the R-tree such as the aR-tree, the aMVR-tree, and the a3DR-tree are employed as the host index and aggregate B-trees (aB-trees) as measure indexes. Given a query, the host index is first searched to identify the set of entries that qualify the spatial condition. The measure indexes of these entries are then accessed to retrieve the timestamps qualifying the temporal conditions.

3. The adaptively partitioned aggregate R-tree

Section 3.1 presents an adaptive partitioning method based on query workloads and *range sum* algorithm of the APART. In [2], we have provided mathematical rationale for the impact of the jurisdiction interval length on the query cost and the index size of the APART. Thus, the detailed description is omitted. Finally, Section 3.2 describes our space-efficient structure for range sum queries with a long time interval.

3.1. Query-adaptive partitioning method

Although spatio-temporal data consist of spatial and temporal information, the time region increases as records are inserted along with time dimension. Therefore, to avoid a large search space, the HR-tree and the MVR-tree maintain multiple R-trees that are created as objects move, while the 3DR-tree does not. Thus, the use of the 3DR-tree and the a3DR-tree suffers from a large search space. To overcome this deficiency of the a3DR-tree, we propose to apply the query-adaptive partitioning method to the a3DR-tree. We call the resultant index structure the *Adaptively Partitioned Aggregate R-Tree (APART)*. The APART consists of a3DR-trees, each of which is responsible for different fixed time interval derived from query workload.

Different from the HR-tree and the MVR-tree, the APART takes advantage of query workloads in order to adaptively partition the time region [2]. The query-adaptive partitioning method can significantly improve the performance of the APART by reducing the search space. Fig. 2 shows an example of an APART that consists of two a3DR-trees (i.e., a3DR-tree [0,5) and a3DR-tree [5,10)). Let a3DR-tree $[t_{st}, t_{ed})$ denote an a3DR-tree whose jurisdiction time interval is fixed to $[t_{st}, t_{ed})$, where t_{st} and t_{ed} are start and end timestamps of the jurisdiction time interval, respectively. A record whose time interval intersects the jurisdiction time interval of an a3DR-tree belongs to the corresponding a3DR-tree. Sometimes, if a record intersects two or more a3DR-trees, it belongs to each of these a3DR-trees simultaneously. Let $rd = \langle s, [t_0, t_1), value\rangle$be a record of an object's movement, where s is the

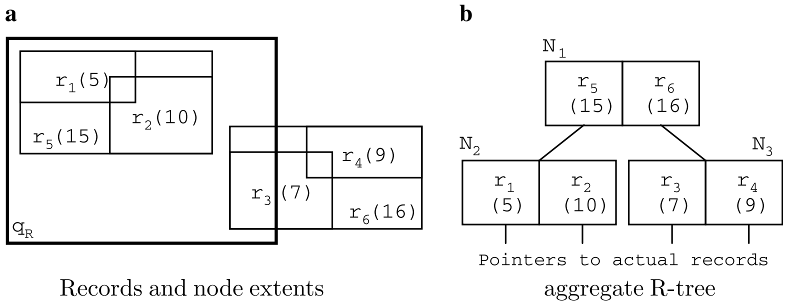

Fig. 1. Example of an aR-tree.

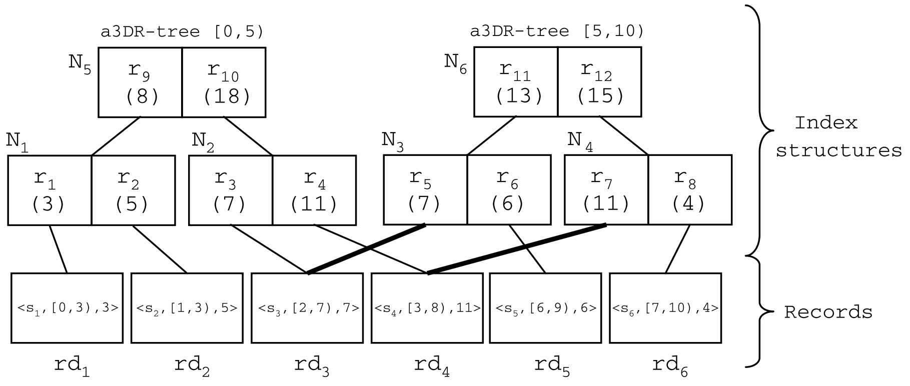

Fig. 2. Partitioning the time region of the a3DR-tree.

spatial area, $[t_0,\ t_1)$ is the time interval, and *value* is the measure attribute of interest which is associated with the spatial area s during the time interval $[t_0, t_1)$. For example, $rd_1 = \langle s_1, [0,3), 3\rangle$belongs to a3DR-tree $[0,5)$ while $rd_3 = \langle s_3, [2,7), 7\rangle$belongs to a3DR-tree $[0,5)$ and a3DR-tree $[5,10)$ simultaneously. In Fig. 2, the two bold lines in a3DR-tree $[5,10)$ are called the duplicate pointers meaning that rd_3 and rd_4 already belong to previous a3DR-tree $[0,5)$.

Let L be the jurisdiction time interval length of a3DR-tree $[t_i,\ t_{i+1})$. Namely, $L = t_{i+1} - t_i$. To determine L dynamically, we take advantage of query workload. Let $\overline{T}_{\text{queries}}$ be the average time interval length of queries and $\overline{T}_{\text{records}}$ be the average time interval length of records. $\overline{T}_{\text{queries}}$ and $\overline{T}_{\text{records}}$ are recomputed whenever queries and records are entered so that they can be adjusted dynamically.

Using $\overline{T}_{\text{queries}}$ and $\overline{T}_{\text{records}}$, the tuning parameter L is dynamically determined by Eq. (1).

$$L = \max(\overline{T}_{\text{queries}}, \overline{T}_{\text{records}}) \tag{1}$$

where $\max(\overline{T}_{\text{queries}}, \overline{T}_{\text{records}})$ chooses the larger value between $\overline{T}_{\text{queries}}$ and $\overline{T}_{\text{records}}$. In Eq. (1), L is automatically determined by using $\overline{T}_{\text{queries}}$ and $\overline{T}_{\text{records}}$ obtained from the query workload and the dataset, respectively. In addition, $L = \max(\overline{T}_{\text{queries}}, \overline{T}_{\text{records}})$ can reflect the changing query workload over time. In [2], we have shown that $L = \max(\overline{T}_{\text{queries}}, \overline{T}_{\text{records}})$ is a good choice in terms of the query cost and the index size. This is confirmed in our experiments.

Fig. 3 presents the range sum algorithm of the APART for computing the sum of values of records that intersect the query window $q = \langle q_{\text{R}}, q_{\text{T}}\rangle$, where q_{R} and q_{T} are the spatial and temporal query windows, respectively. An entry r has the form $\langle r.\text{MBR}, r.\text{lifespan}, r.\text{ptr}, r.\text{aggr_sum}\rangle$where r.MBR and r.lifespan are spatial and temporal extents of the entry, respectively, and r.ptr and r.aggr_sum have the same semantics as the aR-tree. If the time interval which a a3DR-tree is responsible for overlaps the time interval of query, this a3DR-tree is searched to find out records which actually intersect the query window. In the a3DR-tree, the algorithm of computing range sum is the same as that of the aggregate tree structure. (i) If the lifespan of an entry r is covered by q_{T} and the entry's spatial extent is contained in q_{R}, its pre-aggregated sum is used. (ii) The entry's spatial extent partially overlaps q_{R} and its temporal extent also partially overlaps q_{T}. In this case, the algorithm descends to the next R-tree level and the same process is applied recursively, and (iii) if none of the previous conditions holds, the entry is ignored. The final answer is calculated by summing results in each 3DR-tree whose jurisdiction time interval overlaps q_{T}.

Finally, the APART can be incrementally maintained in a straightforward way. If the lifespan of a record does not intersect the boundary, the record is inserted into the a3DR-tree covering the lifespan of the record. Otherwise, the record is shared among multiple a3DR-trees whose jurisdiction intervals intersect the lifespan of the record. The change propagates upward the a3DR-tree, updating the affected entries.

```
func APART_Range_Sum ($A$, $q_R$, $q_T$)
/* $A$ is a pointer to an APART which consists of a set of a3DR-trees,
each of which is responsible for specific time interval.
$q_R$ and $q_T$ are the spatial and temporal query windows, respectively.*/
begin
1. sum := 0
2. for each a3DR-tree $t \in A$ do
3.    if $t$.lifespan $\bigcap q_T \neq \varnothing$ then
4.       sum += a3DR_node_aggregation ($t$.ptr, $q_R$, $q_T$)
5. end-for
6. return sum
end

proc a3DR_node_aggregation ($T_i$, $q_R$, $q_T$)
/* $T_i$ is a pointer to a node of the a3DR-tree.
$q_R$ and $q_T$ are the spatial and temporal query windows, respectively.*/
begin
1. sum := 0
2. for each entry $r \in T_i$ do
3.    if $q_R \bigcap r$.MBR = $r$.MBR and $q_T \bigcap r$.lifespan = $r$.lifespan then
4.       sum += $r$.aggr_sum
5.    else if $q_R \bigcap r$.MBR $\neq \varnothing$ and $q_T \bigcap r$.lifespan $\neq \varnothing$ then
6.       sum += a3DR_node_aggregation($r$.ptr, $q_R$, $q_T$)
7.    else
8.       do nothing
9. end-for
10. return (sum)
end
```

Fig. 3. *Range sum* algorithm.

3.2. Construction of the auxiliary aggregate structure

An auxiliary aggregation structure is used in the APART in order to efficiently process range sum queries with a long time interval. The auxiliary aggregation structure which we call an aggregate spatial and temporal tree (aST-tree) is built on leaf nodes of the APART. Since the number of leaf nodes of the APART is much less than the actual number of records, the size of the aST-tree is expected to be fairly small compared to that of the APART. The aST-tree consists of three kinds of typical aggregate structures as follows: the a3DR-tree, the aggregate B-tree (aB-tree), and the aR-tree. Like the a3DRB-tree, the aST-tree adopts an a3DR-tree as the host index. Specifically, each entry r in the a3DR-tree has the form $\langle r.\text{MBR}, r.\text{lifespan}, r.\text{ptr}, r.\text{aggr\_sum}, r.\text{btree}, r.\text{rtree}\rangle$, where r.MBR, r.lifespan, r.ptr, and r.aggr_sum have the same meaning as the APART, and r.btree points to an aggregate B-tree which stores the detailed measure information of r at concrete timestamps and r.rtree indicates an aggregate R-tree which keeps the detailed measure information of r inside its spatial extent during r.lifespan. Note that its construction is similar to that of the a3DRB-tree except that an additional aR-tree for each entry is utilized to maintain summarized information of records inside r.MBR during its lifespan.

Fig. 4 describes possible relationships between an entry and a query. For convenience, we represent the three dimensional space as the two dimensional space. In this figure, assume that the x and y axes represent spatial and temporal dimensions, respectively. In the case of Fig. 4(a) where the entry is covered by both (q_R and q_T) query extents, the pre-aggregated sum of an entry of the a3DR-tree can answer the range sum query q. In the case of Fig. 4(b) where the entry's spatial extent partially overlaps q_R and its temporal extent is covered by q_T, the aR-tree pointed to by the entry can sufficiently answer q. Conversely, in the case of Fig. 4(c) where the entry's spatial extent is covered by q_R and its temporal extent partially overlaps q_T, the aB-tree pointed to by the entry is sufficient for answering q. In the case of Fig. 4(d) and (e) where the entry is partially covered by both (q_R and q_T) query extents, the algorithm descends to the next a3DR-tree level and the same process is applied recursively.

As with the a3DR-tree, a range sum query is modeled as a 3D box which represents the spatial and temporal ranges. It starts from the root of the 3DR-tree, and each entry r satisfies one of the following conditions: (i) the entry is covered by both (q_R and q_T) query extents. In this case, its pre-computed aggregate sum r.aggr_sum is simply used. (ii) the entry's spatial extent is covered by q_R and its temporal extent partially overlaps q_T. The aB-tree indicated by r.btree is searched to retrieve the range sum for q_T. (iii) the entry's spatial extent partially overlaps q_R and its temporal extent is covered by q_T. In this case, the aR-tree indicated by r.rtree is searched to retrieve range sum for q_R. (iv) if the entry is partially covered by both (q_R and q_T) query extents, the algorithm descends to the next a3DR-tree level and the same process is applied recursively, and (v) if none of the previous conditions holds, the entry is ignored.

Fig. 5 presents the range sum algorithm using the aST-tree. In general, the aST-tree can accelerate the execution of queries under various conditions because (i) if both q_R and q_T are large, many nodes in the intermediate levels of the a3DR-tree are contained in q. As a result, the pre-computed aggregate values of the a3DR-tree are used to answer the range sum query q. (ii) If q_R is small but q_T is large, aR-trees are searched instead of the a3DR-tree. In this case, the pre-computed aggregate values of the aR-trees are used to answer q. (iii) Conversely, if q_R is large but q_T is small, aB-trees are searched instead of the a3DR-tree and the pre-computed aggregate values of the aB-trees are used to answer q. (iv) If both of q_R and q_T are small, the aST-tree behaves as a typical 3DR-tree. In the aST-tree, the summarized information on the spatial and temporal dimensions are kept in separated aggregate indexes, aR-trees and

```
func aST_node_aggregation ($X_i$, $q_R$, $q_T$)
/* $X_i$ is a pointer to a node of the aST-tree. Initially, it points to the root.
$q_R$ and $q_T$ are the spatial and temporal query windows, respectively.*/
begin
1.  sum := 0
2.  for each entry $r \in X_i$ do
3.      if $q_R \bigcap r$.MBR = $r$.MBR and $q_T \bigcap r$.lifespan = $r$.lifespan then
4.          sum += $r$.aggr_sum
5.      else if $q_R \bigcap r$.MBR = $r$.MBR and $q_T \bigcap r$.lifespan $\neq \varnothing$ then
6.          sum += aB_node_aggregation($r$.btree, $q_T$)
7.      else if $q_R \bigcap r$.MBR $\neq \varnothing$ and $q_T \bigcap r$.lifespan = $r$.lifespan then
8.          sum += aR_node_aggregation($r$.rtree, $q_R$)
9.      else if $q_R \bigcap r$.MBR $\neq \varnothing$ and $q_T \bigcap r$.lifespan $\neq \varnothing$ then
10.         sum += aST_node_aggregation ($r$.ptr, $q_R$, $q_T$)
11.     else
12.         do nothing
13. end-for
14. return sum
end
```

Fig. 5. *Range sum* algorithm of the aST-tree.

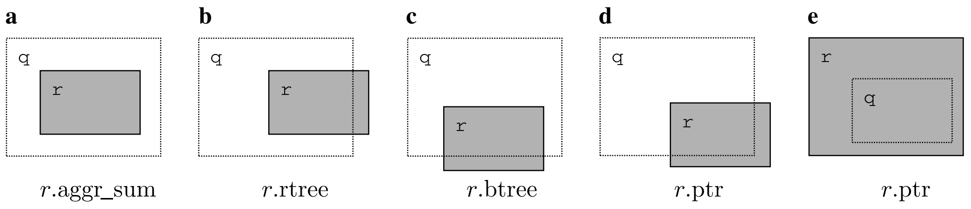

Fig. 4. Relations of entry r to query $q = \langle q_R, q_T\rangle$.

```
func aB_node_aggregation ($B_i$, $q_T$)
/* $B_i$ is a pointer to a node of the aB-tree. Initially it points to the root.
$q_T$ is the temporal query window.*/
begin
1. sum := 0
2. for each entry $b \in B_i$ do
3.     if $q_T \bigcap b$.lifespan = $b$.lifespan then
4.         sum += $b$.aggr_sum
5.     else if $q_T \bigcap b$.lifespan $\neq \varnothing$ then
6.         sum += aB_node_aggregation ($b$.ptr, $q_T$)
7.     else
8.         do nothing
9. end-for
10. return sum
end
```

Fig. 6. *Range sum* algorithm of the aB-tree.

```
func aR_node_aggregation ($R_i$, $q_R$)
/* $R_i$ is a pointer to a node of the aR-tree. Initially, it points to the root.
$q_R$ is the spatial query window.*/
begin
1. sum := 0
2. for each entry $r \in R_i$ do
3.     if $q_R \bigcap r$.MBR = $r$.MBR then
4.         sum += $r$.aggr_sum
5.     else if $q_R \bigcap r$.MBR $\neq \varnothing$ then
6.         sum += aR_node_aggregation ($r$.ptr, $q_R$)
7.     else
8.         do nothing
9. end-for
10. return sum
end
```

Fig. 7. *Range sum* algorithm of the aR-tree.

aB-trees, respectively for each entry of the a3DR-tree used as the host index.

Fig. 6 presents an algorithm for computing the range sum using the aB-tree. The algorithm is straightforward. If the lifespan of an entry b is covered by the time interval of the query q, its pre-aggregated sum is used. If b.lifespan partially overlaps q_T, the algorithm descends to the next aB-tree level and the same process is applied recursively.

Fig. 7 presents an algorithm for computing the range sum using the aR-tree. The algorithm in Fig. 7 is similar to that in Fig. 6. If the spatial extent of an entry r is covered by the spatial window of the query q, its pre-aggregated sum is used. If r.MBR partially overlaps q_R, the algorithm descends to the next aR-tree level and the same process is applied recursively.

Specifically, to create an aST-tree, we should first build the underlying 3DR-tree according to spatial extents and lifespans of leaf nodes in APART. After that, aR-trees and aB-trees of the entries are constructed by scanning the aggregate changes.

4. Experiments

4.1. Experimental environment

In Section 4.2, we experimentally compare our method and its competitor for range sum queries in terms of I/O cost using a system running Windows on a 2.7 GHz processor and 512 MB memory. To represent moving objects, we use Tiger/Line data which represent LA rivers and railways containing 128,971 MBRs [18]. A record consists of three attributes $\langle s, [t_{st}, t_{ed}), value\rangle$, where s is the 2-dimensional spatial area, $[t_{st}, t_{ed})$ is the time interval, and *value* is the measure attribute associated with spatial area s during time interval $[t_{st}, t_{ed})$. The spatial region is a unit square, while the time interval consists of timestamps represented by integers. Objects change their shapes, positions, or values randomly along with time. We investigated their changes for 1000 timestamps. A node corresponds to a page, whose size is set to 4 KB. Let a_{ms} be the ratio of the number of objects that are randomly selected to change their measures at each timestamp and a_{ext} be the ratio of the number of objects that change their regions at each timestamp. For example, $a_{ext} = 5\%$ means that 6448 $(= 128{,}971 \times 5\%)$ objects change their shapes or positions at each timestamp. Each query specifies a square spatial region with side length q_S and a temporal interval including q_T timestamps. To investigate the performance of range sum queries, we execute workloads of 200 queries whose locations are randomly selected in the whole region.

Note that in this work, the implementation of all the spatio-temporal aggregate index structures is based on the algorithms of the R*-tree [1]. In [15], Tao and Papadias stated that the a3DRB-tree has the best performance for volatile regions and the smallest size. Therefore, we compared the APART with the a3DRB-tree to prove the superiority of our approach. In these experiments, we fix L to $\overline{T}_{records}$ since query workloads are not known a-priori while building the APART.

4.2. Experimental results

We compare the APART and its competitor a3DRB-tree in terms of the size and the query costs. Fig. 8(a) shows the sizes of two aggregation structures as a function of a_{ms} by fixing a_{ext} to 5% and Fig. 8(b) plots the sizes as a function of a_{ext} by fixing a_{ms} to 10%. Note that a3DRB-trees are the smallest in all cases because they do not incur redundancy. The size of an APART is larger than that of the a3DRB-tree due to the data duplication introduced by the partitioning method.

Fig. 9 presents query performance of methods for volatile regions. In Fig. 9(a), we measure the query cost as a function of q_S. As expected, the APART performs better than the a3DRB-tree. The reason is that in the aST-tree, the summarized information on the spatial and temporal dimensions are kept in two kinds of separated indexes, aR-trees and aB-trees, respectively for each entry of the a3DR-tree. Fig. 9(b) shows the query cost as a function of q_T by fixing q_S, a_{ms}, and a_{ext} to 0.3, 10%, 5%, respectively. The APART outperforms the a3DRB-tree for time interval queries since aR-trees are visited to answer these queries. Fig. 9(c) shows the query cost as a function of a_{ms}. The costs of the APART and the a3DRB-tree are not affected at all since the B-tree height of each entry does not change. Fig. 9(d) presents the cost as a function of a_{ext}.

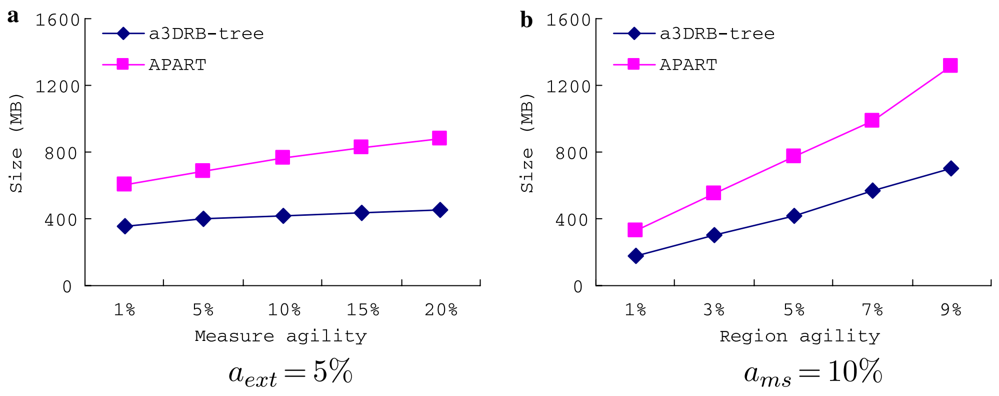

Fig. 8. Size comparison.

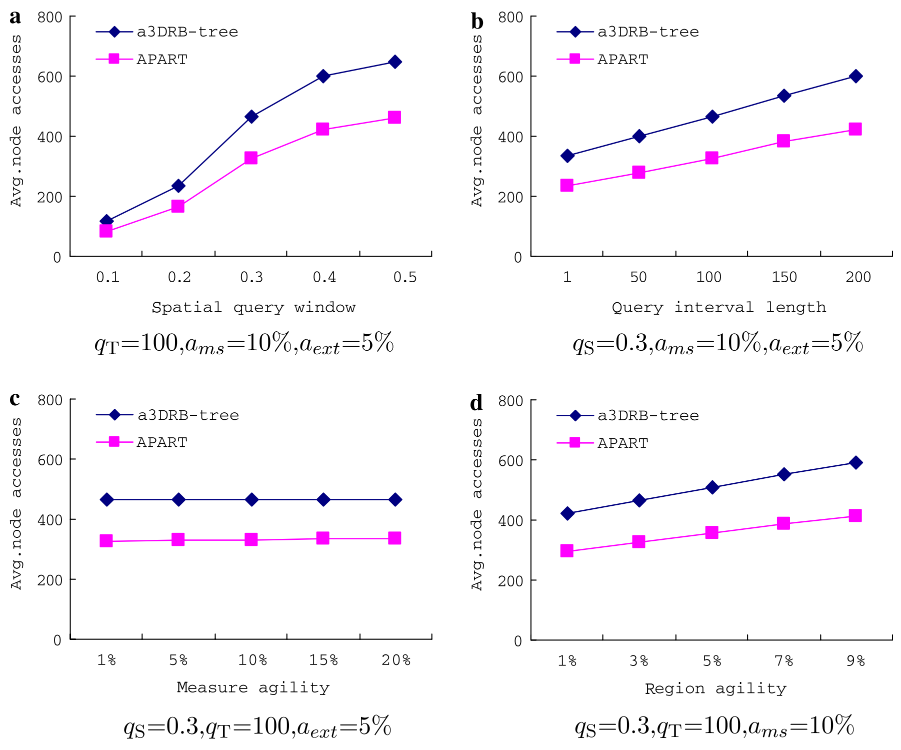

Fig. 9. Comparison of query performance.

The query cost using the APART is reduced because the APART employs an aR-tree within each entry for keeping the summarized information on spatial dimension.

5. Conclusions

With a rapid development of mobile, satellite, and wireless technologies, numerous real-life applications (e.g., traffic control systems or mobile communication systems) that require range sum queries have appeared in spatio-temporal databases. Unlike other aggregate structures, the APART employs a query-adaptive partitioning method that plays an important role in achieving the best performance for various query workloads and datasets. With regard to the structure built already, in order to efficiently process the changed query workload using the past structure, we add to the APART an auxiliary aggregate index called the aST-tree, which maintains summarized information on the spatial and temporal dimensions in two kinds of separated indexes, aR-trees and aB-trees, respectively. Through several experiments that simulate real-life situations, we showed that the APART is superior to the a3DRB-tree in a wide range of scenarios.

We believe that the range sum query to moving objects is a promising and challenging research area from the practical as well as theoretical point of view. Sometimes, depending on applications where fast response time is more crucial than exact query results, approximate query results with fast response time may be required rather than exact query results with unacceptable response time. To satisfy such requirements, instead of the range sum algorithm presented in this paper, the progressive approximate algorithm [7] can be adopted by the APART.

Acknowledgment

This research was supported by the Agency for Defense Development, Korea, through the Image Information Research Center at Korea Advanced Institute of Science & Technology.

References

[1] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles, in: Proceedings of the ACM SIGMOD Conference on Management of Data, 1990, pp. 322–331.
[2] H. Cho, J. Min, C. Chung, An adaptive indexing technique using spatio-temporal query workloads, Inform. Software Tech. 46 (4) (2004) 229–241.
[3] J. Gray, A. Bosworth, A. Layman, H. Pirahesh, Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total, in: Proceedings of International Conference on Data Engineering, 1996, pp. 152–159.
[4] A. Guttman, R-Trees: a dynamic index structure for spatial searching, in: Proceedings of the ACM SIGMOD Conference on Management of Data, 1984, pp. 47–57.
[5] V. Harinarayan, A. Rajaraman, J. Ullman, Implementing data cubes efficiently, in: Proceedings of the ACM SIGMOD Conference on Management of Data, 1996, pp. 205–216.
[6] M. Kolahdouzan, C. Shahabi, Voronoi-based K nearest neighbor search for spatial network databases, in: Proceedings of the International Conference on Very Large Data Bases, 2004, pp. 840–851.
[7] I. Lazaridis, S. Mehrotra, Progressive approximate aggregate queries with a multi-resolution tree structure, in: Proceedings of the ACM SIGMOD Conference on Management of Data, 2001, pp. 401–412.
[8] M. Nascimento, J. Silva, Towards historical R-trees, in: Proceedings of the ACM Symposium on Applied Computing, 1998, pp. 235–240.
[10] D. Papadias, P. Kalnis, J. Zhang, Y. Tao, Efficient OLAP operations in spatial data warehouses, in: Proceedings of International Symposium on Spatial and Temporal Databases, 2001, pp. 443–459.
[11] D. Pfoser, C. Jensen, Y. Theodoridis, Novel approaches in query processing for moving object trajectories, in: Proceedings of the International Conference on Very Large Data Bases, 2000, pp. 395–406.
[12] S. Saltenis, C. Jensen, S. Leutenegger, M. Lopez, Indexing the positions of continuously moving objects, in: Proceedings of the ACM SIGMOD Conference on Management of Data, 2000, pp. 331–342.
[13] Z. Song, N. Roussopoulos, k-nearest neighbor search for moving query point, in: Proceedings of International Symposium on Spatial and Temporal Databases, 2001, pp. 79–96.
[14] Y. Tao, D. Papadias, The MV3R-Tree: a spatio-temporal access method for timestamp and interval queries, in: Proceedings of the International Conference on Very Large Data Bases, 2001, pp. 431–440.
[15] Y. Tao, D. Papadias, Historical spatio-temporal aggregation, ACM Trans. Inform. Syst. 23 (1) (2005) 61–102.
[17] M. Vazirgiannis, Y. Theodoridis, T. Sellis, Spatio-temporal composition and indexing for large multimedia applications, Multimedia Syst. 6 (4) (1998) 284–298.
[18] US Bureau of the Census: Technical Documentation, TIGER/Line Files, 1995.

Further reading

[1] M. Nascimento, J. Silva, Y. Theodoridis, Evaluation of access structures for discretely moving points, in: Proceedings of International Workshop on Spatio-Temporal Database Management, 1999, pp. 171–188.
[2] Y. Theodoridis, J. Silva, M. Nascimento, On the generation of spatio-temporal datasets, in: Proceedings of International Symposium on Large Spatial Databases, 1999, pp. 147–164.