

Historical Spatio-Temporal Aggregation

YUFEI TAO

City University of Hong Kong, Hong Kong, China

DIMITRIS PAPADIAS

Hong Kong University of Science and Technology, Hong Kong, China

Spatio-temporal databases store information about the positions of individual objects over time. However, in many applications such as traffic supervision or mobile communication systems, only summarized data, like the number of cars in an area for a specific period, or phone-calls serviced by a cell each day, is required. Although this information can be obtained from operational databases, its computation is expensive, rendering online processing inapplicable. In this paper, we present specialized methods, which integrate spatio-temporal indexing with pre-aggregation. The methods support dynamic spatio-temporal dimensions for the efficient processing of historical aggregate queries without a-priori knowledge of grouping hierarchies. The superiority of the proposed techniques over existing methods is demonstrated through a comprehensive probabilistic analysis and an extensive experimental evaluation.

Categories and Subject Descriptors: H.2 [Database Management]; H3.3 [Information Storage and Retrieval]

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Aggregation, Access Methods, Cost Models

1. INTRODUCTION

Spatio-temporal databases have received considerable attention during the past few years due to the accumulation of large amounts of multi-dimensional data evolving in time, and the emergence of novel applications such as traffic supervision and mobile communication systems. Research has focused on modeling [Sistla et al. 1997, Güting et al. 2000, Forlizzi et al. 2000], historical information retrieval [Vazirgiannis et al. 1998, Pfooser et al. 2000, Kollios et al. 2001, Tao and Papadias 2001], indexing of moving objects [Kollios et al. 1999, Agarwal et al. 2000, Saltenis et al. 2000, Hadjieleftheriou et al. 2002, Saltenis and Jensen 2002, Tao et al. 2003a], selectivity estimation [Choi and Chung 2002, Hadjieleftheriou et al. 2003, Tao et al. 2003b], etc. All these approaches assume that object locations are individually stored, and queries retrieve objects that satisfy some spatio-temporal condition (e.g., mobile users inside a query window during a time interval, or the first car expected to arrive at a destination, etc.).

The motivation of this work is that many (if not most) current spatio-temporal applications require summarized results, rather than information about individual objects. As an example, traffic supervision systems monitor the number of cars in an area of interest [Denny et al. 2003], instead of their ids. Similarly mobile phone companies use the number of phone-calls per cell in order to identify trends and prevent potential network congestion. Other applications focus directly on numerical aggregate data with spatial and temporal aspects, rather than moving objects. As an example consider a pollution monitoring system, where the readings from several sensors are fed into a database that arranges them in regions of similar or identical values. These regions should then be indexed for the

This research was supported by the grants CityU 1163/04E and HKUST 6197/02E from Hong Kong RGC.

Authors' addresses: Yufei Tao, Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong; email: taoyf@cs.cityu.edu.hk; Dimitris Papadias, Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong; email: dimitris@cs.ust.hk.

This is a preliminary release of an article accepted by ACM Transactions on Information Systems. The definitive version is currently in production at ACM and, when released, will supercede this version. Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

efficient processing of queries such as "find the areas near the center with the highest pollution levels yesterday".

Although summarized results can be obtained using conventional operations on individual objects (i.e., accessing every single record qualifying the query), the ability to manipulate aggregate information *directly* is imperative in spatio-temporal databases due to several reasons. First, in some cases personal data should not be stored due to legal issues. For instance, keeping historical locations of mobile phone users may violate their privacy. Second, the individual data may be irrelevant or unavailable, as in the traffic supervision system mentioned above. Third, although individual data may be highly volatile and involve extreme space requirements, the aggregate information usually remains fairly constant for long periods, thus requiring considerably less space for storage. For example, although the distinct cars in a city area usually change rapidly, their number at each timestamp may not vary significantly, since the number of objects entering the area is similar to that exiting. This is especially true if only approximate information is kept, i.e., instead of the precise number of objects we store values to denote ranges such as high or low traffic etc.

We consider, at the finest aggregation unit, a set of regions that can be static (e.g., road segments), or volatile (e.g., areas covered by antenna cells, which can change their extents according to the weather conditions, allocated capacity, etc.). Each region is associated with a set of measures (e.g., number of cars in a road segment, phone-calls per cell), whose values are continuously updated. We aim at retrieving aggregate measures over regions satisfying certain spatio-temporal conditions, e.g., "return the number of cars in the city center during the last hour" (a formal problem definition is presented in Section 3). An important fact that differentiates spatio-temporal from conventional aggregation is the lack of pre-defined groupings on the aggregation units. Such groupings (e.g., product types) are taken into account in traditional data warehouses so that queries of the form "find the average sales for all products grouped-by product type" can be efficiently answered. In spatio-temporal scenarios, the spatial and temporal extents of queries do not confine to pre-defined groupings, and cannot be predicted (e.g., queries can inquire about the traffic situation in any district of arbitrary size at any time interval).

This paper presents several multi-tree indexes that combine the spatial and temporal attributes to accelerate query processing involving static or volatile spatial dimensions. The proposed indexes support ad-hoc groupings, arbitrary query windows and historical time intervals. Furthermore, we perform a comprehensive analysis for the existing and proposed solutions, which provides significant insight into their behavior and reveals the superiority of our methods. This analysis leads to a set of cost models directly applicable for query optimization in practice. The rest of the paper is organized as follows. Section 2 describes related work in the context of spatial, spatio-temporal databases and conventional data warehouses. Section 3 formally describes the problem and elaborates its characteristics. Section 4 presents the proposed solutions, while Section 5 analyzes their performance. Section 6 contains an extensive experimental evaluation and Section 7 concludes the paper with a discussion on future work.

2. RELATED WORK

Section 2.1 introduces the spatial and spatio-temporal indexes fundamental to our discussions. Then, Section 2.2 surveys existing techniques for multi-dimensional aggregate processing and Section 2.3 reviews traditional data

warehouses and their extensions for spatio-temporal data.

2.1 Spatial and spatio-temporal access methods

Spatial access methods [Gaede and Günther 1998] manage multi-dimensional (typically 2D or 3D) rectangles, and are often optimized for the *window query*, which retrieves the objects intersecting a query box. One of the most popular indexes is the R-tree [Guttman 1984] and its variations, most notably the R*-tree [Beckmann et al. 1990]. Each intermediate entry r of an R-tree has the form $\langle r.MBR, r.pointer \rangle$, where $r.MBR$ is the minimum bounding rectangle that tightly encloses all objects in its sub-tree pointed to by $r.pointer$. For leaf entries, $r.MBR$ stores the corresponding data rectangle whose actual record is referenced by $r.pointer$. Figure 1a illustrates four 2D rectangles R_1, \dots, R_4 , together with the node MBRs of the corresponding R-tree (node capacity=2) shown in Figure 1b. Based on their spatial proximity, R_1, R_2 are grouped together into node N_1 (whose parent entry is R_5) and R_3, R_4 into N_2 (parent entry R_6). Given a window query q_R (e.g., the grey rectangle in Figure 1a), the qualifying objects (i.e., R_1, R_2, R_3) are retrieved by visiting those nodes whose MBRs intersect q_R .

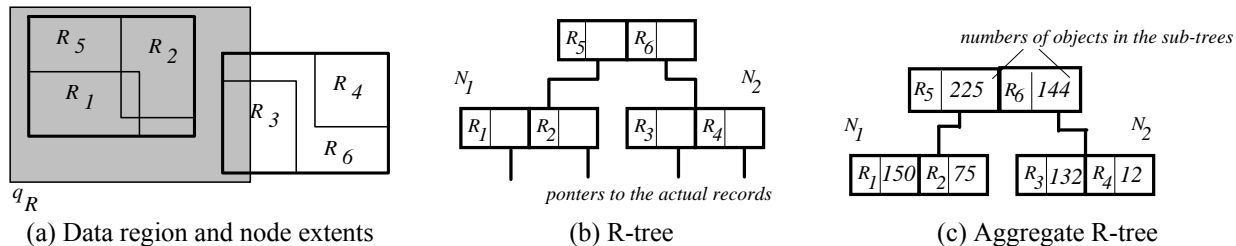


Fig.1. R-tree example

A spatio-temporal index, on the other hand, manages moving objects. In [Vazirgiannis et al. 1998] the movements of 2D rectangles are modeled as 3D boxes indexed with a *3DR-tree*. Specifically, the temporal projection of a box denotes the period when the corresponding object remains static, while the spatial projection corresponds to the object's position and extents during that period. Whenever an object moves to another position, a new box is created to represent its new static period, position, and extents. A *spatio-temporal window query* involves, in addition to a spatial region q_R , a time interval q_T , and returns objects intersecting q_R during q_T . If we model the query also as a 3D box (bounding q_R and q_T), the qualifying objects are those whose 3D representation intersects the query box. A similar idea is applied in [Pfoser et al. 2000] for storing objects' trajectories.

While the *3DR-tree* stores all data versions in a single tree, the *partially persistent* technique [Becker et al. 1996, Varman and Verma 1997, Salzberg and Tsotras 1999] maintains (in a space efficient manner) a separate (logical) 2D R-tree for each timestamp, indexing the regions that are alive at this timestamp. The motivation is that the number of records valid at a timestamp is much lower than the total number of data versions in history; hence, a query with short interval (compared to the history length) only needs to search a small number of R-trees, each indexing a limited number of objects. A popular index is the *Multi-version R-tree (MVR-tree)* [Kumar et al. 1998, Tao and Papadias 2001]. An entry r has the form $\langle r.MBR, r.t_{st}, r.t_{ed}, r.pointer \rangle$, where $[r.t_{st}, r.t_{ed}]$ denotes the *lifespan*, i.e., the time interval during which r was alive ($t_{ed} = "*" implies that the entry is still alive at the current time). For leaf entries, $r.MBR$ denotes the MBR of the corresponding object, while for intermediate entries it encloses all the child entries alive in its lifespan. The semantics of $r.pointer$ are similar to the ordinary R-tree.$

Figure 2a shows an example where R_1 moves to a new position R_1' at timestamp 5 (triggering the change of the parent entry R_5 to R_5'), and Figure 2b illustrates the corresponding *MVR-tree*. The (logical) R-trees for time interval $[1,4]$ involve entries in nodes N_1, N_2, N_4 (observe the lifespans of their parent entries), while starting from timestamp 5, the logical trees consist of nodes N_5 and N_3 , which replace N_4 and N_1 , respectively. Note that N_2 is shared (i.e., it is the child node of both N_4 and N_5) because none of its objects issued an update. The window query algorithm of the *MVR-tree* is the same as that of normal R-trees, except that search is performed in the logical trees responsible for the query timestamps. If the number of involved timestamps is small, only few R-trees are accessed, in which case the *MVR-tree* is more efficient than the *3DR-tree*. This benefit comes, however, at the cost of data duplication. In Figure 2, for example, although region R_2 does not issue any update, two separate copies R_2, R_2' are stored in N_1, N_3 respectively. As a result, the *MVR-tree* performs worse for queries involving long temporal intervals and consumes more space than the corresponding *3DR-tree* [Tao and Papadias 2001].

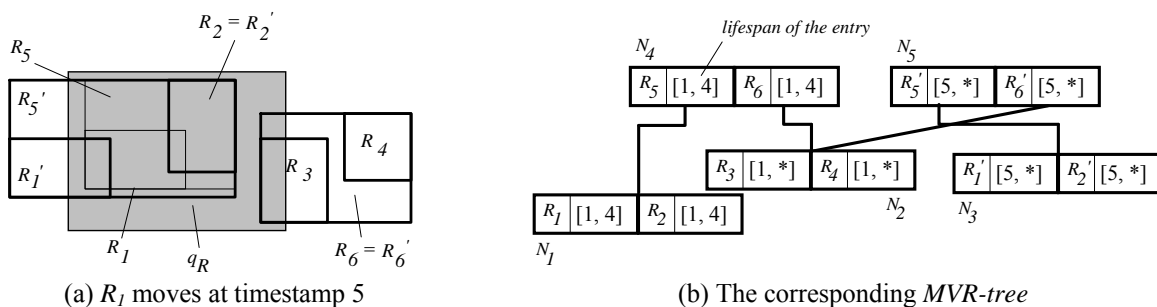


Fig.2. The multi-version R-tree

2.2 Multi-dimensional aggregate methods

The *aggregate R-tree* (aR-tree) [Jurgens and Lenz 1998, Papadias et al. 2001] augments traditional R-trees with summarized information. Figure 1c shows an example aR-tree for the regions of Figure 1a. Each leaf entry contains a set of numerical *measures*, which are the objectives of analysis (e.g., the number of users in a cell, the number of phone call made). The measures of intermediate entries are computed using some distributive¹ aggregation function (e.g., *sum*, *count*, *max*), and summarize the information in the corresponding subtrees. In Figure 1c we assume that there is a single measure per leaf entry (i.e., data region); the measure for intermediate entries is based on the *sum* function, i.e., the measure of entry R_5 equals the sum of measures of R_1 and R_2 (e.g., the total number of users in the regions indexed by its subtree). The same concept has been applied to a variety of indexes [Lazaridis and Mehrotra 2001].

The aR-tree (and other multi-dimensional aggregation structures) aims at the efficient processing of the *window aggregate query*. Such a query specifies a window q_R and returns the aggregated measure of the regions intersecting q_R (instead of reporting them individually). For instance, if the query window q_R of Figure 1a is applied to the aR-tree of Figure 1c, the result should be $150+75+132$ (i.e., the sum of measures of regions R_1, R_2, R_3). Since R_5 .MBR is covered by q_R , all the objects in its sub-tree must satisfy the query. Thus, the measure (225) stored with R_5 is

¹ A function f_{agg} is distributive [Gray et al. 1996] if, given $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$, $f_{agg}(S)$ can be obtained from $f_{agg}(S_1)$ and $f_{agg}(S_2)$, namely, the aggregate result for S can be computed by further aggregating disjoint subsets S_1, S_2 .

aggregated directly, without accessing its child node. On the other hand, since R_6 .MBR partially intersects q , its subtree must be visited to identify the qualifying regions (only R_3). Hence, the query is answered with only 2 node accesses (root and N_2), while a traditional R-tree requires 3 accesses.

Multi-dimensional aggregate processing has also been studied theoretically, leading to several interesting results. Zhang et al. [Zhang et al. 2001] propose the *MVSB-tree* that efficiently solves a window aggregate query on two-dimensional horizontal interval data (i.e., find the number of intervals intersecting a query window) in $O(\log_B(N/B))$ I/Os using $O((N/B)\log_B(N/B))$ space, where N is the dataset cardinality and B the disk page capacity. Their idea is to transform a query to four “less-key-less-time” and two “less-key-single-time” queries, which are supported by two separate structures that constitute a complete *MVSB-tree*. This solution also answers aggregate queries on 2D points (e.g., find the number of points in a query window) with the same performance by treating each point as a special interval with zero length. The *aP-tree* [Tao et al. 2002b] achieves the same time and space complexity using a simpler conversion of a window aggregate query to two “vertical range queries”. Govindarajan, et al. [Govindarajan et al. 2003] present the *CRB-tree* that further lowers the space consumption, and supports data points of arbitrary dimensionality.

The above techniques target point/interval objects (they are inapplicable to regions), while Zhang et al. [Zhang et al. 2002] develop two versions of the *ECDF-B-tree* for answering aggregate queries on rectangular data with different space-query time tradeoffs. Specifically, in the d -dimensional space, the first version consumes $O((N/B)\log_B^{d-1}(N/B))$ space and answers a query in $O(B\log_B^d(N/B))$ I/Os, while the corresponding complexities of the second version are $O(N\cdot B^{d-2}\log_B^{d-1}(N/B))$ (for space) and $O(\log_B^d(N/B))$ (for query cost). Both versions, however, require relatively high space consumption, limiting their applicability in practice. Aggregate processing on one-dimensional intervals has also been addressed in the context of temporal databases [Kline and Snodgrass 1995, Gendrano et al. 1999, Moon et al. 2000, Yang and Widom 2003]. Zhang et al. [Zhang et al. 2002, Zhang et al. 2003] study spatial and temporal aggregation over data streams.

2.3 Data warehouses

A considerable amount of related research has been carried out on data warehouses and OLAP in the context of relational databases. The most common conceptual model for data warehouses is the multi-dimensional data view. In this model, each measure depends on a set of dimensions, e.g., *region* and *time*, and thus is a value in the multi-dimensional space. A dimension is described by a domain of values (e.g. days), which may be related via a hierarchy (e.g., day-month-year). Figure 3 illustrates a simple case, where each cell denotes the measure of a region at a certain timestamp. Observe that although regions are 2-dimensional, they are mapped as one dimension in the warehouse.

The star schema [Kimball 1996] is a common way to map a data warehouse onto a relational database. A main table (called *fact table*) F stores the multi-dimensional array of measures, while auxiliary tables D_1, D_2, \dots, D_n store the details of the dimensions. A tuple in F has the form $\langle D_i[.key], M[\] \rangle$ where $D_i[.key]$ is the set of foreign keys to the dimension tables and $M[\]$ is the set of measures. OLAP operations ask for a set of tuples in F , or for aggregates on groupings of tuples. Assuming that there is no hierarchy in the dimensions of the previous example, the possible

groupings in Figure 3 include: (i) group-by Region and Time, which is identical to F , (ii)-(iii) group-by Region (Time), which corresponds to the projection of F on the *region-* (*time-*) axis, and (iv) the aggregation over all values of F which is the projection on the origin (Figure 3 depicts these groupings for the aggregation function *sum*). The fact table together with all possible combinations of group-bys composes the *data cube* [Gray et al. 1996]. Although all groupings can be derived from F , in order to accelerate query processing some results may be pre-computed and stored as *materialized views*.

	aggregate results over timestamps					total sum
	369	369	367	364	359	1828
R_4	12	12	12	12	12	60
R_3	132	127	125	127	127	638
R_2	75	80	85	90	90	420
R_1	150	150	145	135	130	710
	T_1	T_2	T_3	T_4	T_5	aggregate results over regions
	time					

FACT TABLE

Fig.3. A data cube example

A detailed group-by query can be used to answer more abstract aggregates. In our example, the total measure of all regions for all timestamps (i.e. 1828) can be computed either from the fact table, or by summing the projected results on the *time* or *region* axis. Ideally the whole data cube should be materialized to enable efficient query processing. Materializing all possible results may be prohibitive in practice as there are $O(2^n)$ group-by combinations for a data warehouse with n dimensional attributes. Therefore, several techniques have been proposed for the view selection problem in OLAP applications [Harinarayan et al. 1996, Gupta 1997, Gupta and Mumick 1999, Baralis et al. 1997, Shukla et al. 1998]. In addition to relational databases, data warehouse techniques have also been applied to spatial [Han et al. 1998, Stefanovic et al. 2000] and temporal [Mendelzon and Vaisman 2000, Hurtado et al. 1999] databases. All these methods, however, benefit only queries on a predefined hierarchy. An ad-hoc query not confined by the hierarchy, such as the one in Figure 3 involving the gray cells, would still need to access the fact table, even if the entire data cube were materialized. In the next section we formally define spatio-temporal aggregate processing and explain the inefficiency of existing techniques.

3. PROBLEM DEFINITION AND CHARACTERISTICS

Consider N regions R_1, R_2, \dots, R_N , and a time axis consisting of discrete timestamps $1, 2, \dots, T$, where T represents the total number of recorded timestamps (i.e., the length of history). Following the conventional spatial object modeling, each region R_i ($1 \leq i \leq N$) is a two-dimensional minimum bounding rectangle of the actual shape (e.g., a road segment, an antenna cell, etc). The position and area of a region R_i may vary along with time, and we refer to its extent at timestamp t as $R_i(t)$. Each region carries a set of measures $R_i(t).ms$, which also changes with time (sometimes we refer to $R_i(t).ms$ as the *aggregate data* of $R_i(t)$). Note that this modeling trivially captures static objects, for which $R_i(t)$ remains constant for all timestamps t . Further, it also supports region insertions/deletions, i.e., the

emergence/disappearance of new/existing objects. In this case, the dataset cardinality N should be interpreted as the total number of *distinct* regions in the entire history. At a timestamp t , if a region R_i ($1 \leq i \leq N$) is inactive (i.e., it has been deleted or has not been inserted at this time), its extent $R_i(t)$ and measure $R_i(t).ms$ are set to some default “void” values. Without loss of generality, to simplify discussion in the sequel we do not consider such appearances/disappearances, and assume that N regions are active at *all* timestamps.

In practice the measures of regions change asynchronously with their extents. In other words, the measure of R_i ($1 \leq i \leq N$) may change at a timestamp t (i.e., $R_i(t).ms \neq R_i(t-1).ms$), while its extent remains the same (i.e., $R_i(t) = R_i(t-1)$), and vice versa. To quantify the rates of these changes, we define the *measure agility* $a_{ms}(t)$, as the percentage of regions that issue measure modifications at time t (e.g., if $a_{ms} = 100\%$, then all regions obtain new measures each timestamp); similarly, the *extent agility* $a_{ext}(t)$ characterizes the percentage for extent changes. In some cases the extent agility is 0 (e.g., road segments are static). Even for volatile regions (i.e., $a_{ext}(t) > 0$), $a_{ms}(t)$ is *usually considerably higher than* $a_{ext}(t)$, which is an important property that must be taken into account for efficient query processing.

We aim at answering the *spatio-temporal window aggregate* query, which specifies a rectangle q_R and a time interval q_T of continuous timestamps. The goal is to return the aggregated measure $Agg(q_R, q_T, f_{agg})$ of all regions that intersect q_R during q_T , according to some distributive aggregation function f_{agg} , or formally:

$$Agg(q_R, q_T, f_{agg}) = f_{agg} \{ R_i(t).ms \mid R_i(t) \text{ intersects } q_R \text{ and } t \in q_T \}.$$

If q_T involves a single timestamp, the query is a *timestamp query*; otherwise, it is an *interval query*. For the following examples and illustrations, we use the static (dynamic) regions of Figure 1 (2), assuming that a region corresponds to the coverage area of an antenna cell. For each data region $R_i(t)$ there is a single measure $R_i(t).ms$ (we use the measures of Figure 3) representing the number of phone-calls initialized in R_i at timestamp t and the aggregate function is *sum*. A spatio-temporal window aggregate query (q_R, q_T) retrieves the total number of phone-calls initiated during q_T in cells intersecting q_R . Application to other aggregate functions and query types is, as discussed in Section 7, straightforward. Next, we describe how to adapt existing methods to spatio-temporal aggregation, and explain their inefficiency.

- **Using a 3D aggregate R-tree**

We can consider the problem as multi-dimensional aggregate retrieval in the 3D space and solve it using one of the existing aggregation structures (discussed in Section 2.2). Assume for instance that we use aR-trees. Whenever the extent or measure of a region changes, a new 3D box is inserted in a 3D version of the aR-tree, called the *a3DR-tree*. Using the example of Figure 3, four entries are required for R_i : one for timestamps 1 and 2 (when its measure remains 150) and three more entries for the other timestamps. Given a spatio-temporal window aggregate query, we can also model it as a 3D box, which can be processed in a way similar to Figure 1c. The problem of this solution is that it creates a new box duplicating the region’s extent, even though it does not change. Since the measure changes are much more frequent than extent updates, the *a3DR-tree* incurs high redundancy. The worst case occurs when $a_{ext}(t) = 0$: although the extent of a region remains constant, it is still duplicated at the rate of its measure changes. Bundling the extent and aggregate information in all entries significantly lowers the node fanout and compromises

query efficiency, because as analyzed in Section 5, more nodes must be accessed to retrieve the same amount of information. Note that redundancy incurs *whenever the extent and measure changes are asynchronous*, i.e., the above problem also exists when a new box is spawned because of an extent update, in which case the region's measure must be replicated.

- **Using a data cube**

Following the traditional data warehouse approach we could create a data cube, where one axis corresponds to time, the other to regions, and keep the measure values in the cells of this two-dimensional table (see Figure 3). Since the spatial dimension has no one-dimensional order we store the table in the secondary memory ordered by time and build a B-tree index to locate the pages containing information about each timestamp. The processing of a query employs the B-tree index to retrieve the pages (i.e., table columns) containing information about q_T ; then, these regions (qualifying the temporal condition) are scanned sequentially and the measures of those satisfying q_R are aggregated. In the sequel, we refer to this method as *column scanning*.

Even if there exists an additional spatial index on the regions, the simultaneous employment of both indexes has limited effect. Assume that first a window query q_R is performed on the spatial index to provide a set of ids for regions that qualify the spatial condition. Measures of these regions must still be retrieved from the columns corresponding to q_T (which, again, are found through the B-tree index). However, the column storage does not preserve spatial proximity, and hence the spatially qualifying regions are expected to be scattered in different pages. Therefore, the spatial index has some effect only on very selective queries (on the spatial conditions). Furthermore, recall that pre-materialization is useless, since the query parameters q_R and q_T do not conform to pre-defined groupings.

4. PROPOSED SOLUTIONS

Our solutions are motivated by the facts that (i) the extent and measure updates are asynchronous and (ii) in practice, measures change much more frequently than extents (which may be even static). Therefore, the two types of updates should be managed independently to avoid redundancy. In particular, the proposed solutions involve 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. Figure 4 shows a general overview of the architecture. Given a query, the host index is first searched, identifying 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. Since the number of records (corresponding to extent changes) in the host index is very small compared to the measure changes, the cost of query processing is expected to be low. As host indexes we use variations of the R-tree due to its popularity, flexibility (i.e., applicability to spatial or spatio-temporal data), low space consumption ($O(N/B)$) and good performance in practice. For similar reasons, we use aggregate B-trees as measure indexes. Nevertheless, the same concept can be applied with other spatial or temporal aggregate structures. In Section 4.1, we first solve the case of static regions (i.e., $a_{ext}(t)=0$). Then, Sections 4.2 and 4.3 address the general problem involving volatile regions

($a_{ext}(t) > 0$). Section 4.4 proposes a space-efficient structure for managing multiple measure indexes.

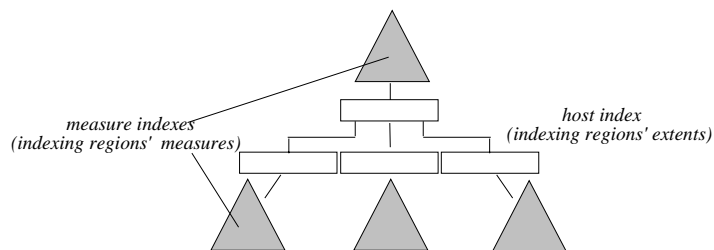


Fig.4. Overview of the proposed solution

4.1 The aggregate R-B-tree

The *aggregate R- B-tree (aRB-tree)* adopts an aR-tree as the host index, where an entry r has the form $\langle r.MBR, r.aggr, r.pointer, r.btree \rangle$; $r.MBR$ and $r.pointer$ have the same semantics as a normal R-tree, $r.aggr$ keeps the aggregated measure about r over the *entire history*, and $r.btree$ points to an aggregate B-tree which stores the detailed measure information of r at concrete timestamps. Figure 5 illustrates an example using the data regions of Figure 1a and the measures of Figure 3. The number 710 stored with R-tree entry R_1 , equals the sum of measures in R_1 for all 5 timestamps (e.g., the total number of phone calls initiated at R_1). The first leaf entry of the B-tree for R_1 (1, 150) indicates that the measure of R_1 at timestamp 1 is 150. Since the measure of R_1 at timestamp 2 is the same, there is no a special entry, but this knowledge is implied from the previous entry (1, 150). Similarly, the first root entry (1, 445) of the same B-tree indicates that the aggregated measure in R_1 during time interval [1,3] is 445. The topmost B-tree stores aggregated information about the whole space, and its role is to answer queries involving only temporal conditions (similar to that of the extra row in Figure 3).

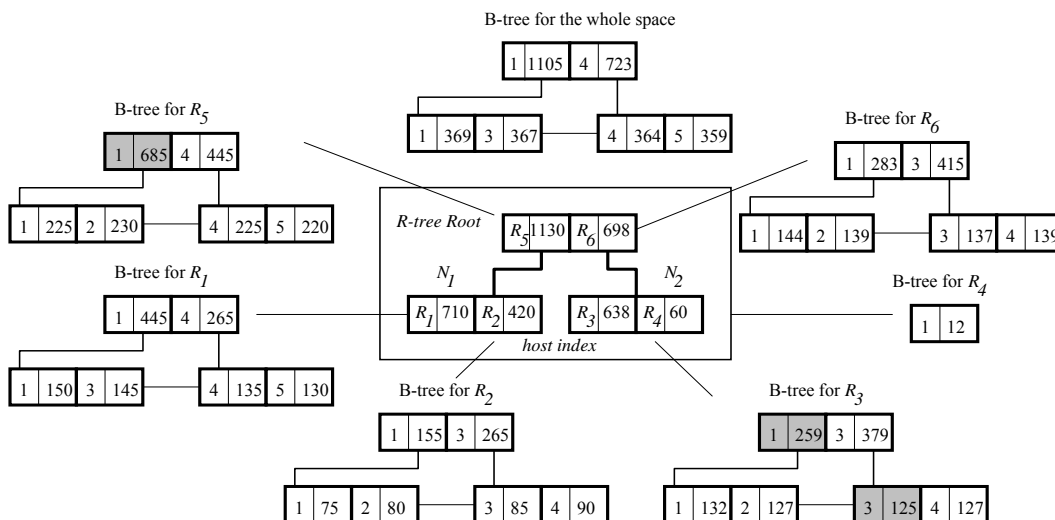


Fig.5. An *aRB-tree* (c.f. regions in Figure 1a and measures in Figure 3)

To illustrate the processing algorithms, consider the query "find the number of phone-calls initiated during interval $q_T=[1,3]$ in all cells intersecting the window q_R shown in Figure 1a". Starting from the root of the R-tree, the algorithm visits the B-tree of R_5 since the entry is totally contained in q_R . The root of this B-tree has entries (1,685), (4,445) meaning that the aggregated measures (of all data regions covered by R_5) during intervals [1,3], [4,5] are 685

and 445, respectively. Hence the contribution of R_5 to the query result is 685. The second root entry R_6 of the R-tree partially overlaps q_R , so we visit its child node, where only entry R_3 intersects q_R , and thus its B-tree is retrieved. The first entry of the root (of the B-tree) suggests that the contribution of R_3 for the interval [1,2] is 259. In order to complete the result we will have to descend the second entry and retrieve the measure of R_3 at timestamp 3 (i.e., 125). The final result equals 685+259+125, which corresponds to the sum of measures in the gray cells of Figure 5.

The pseudo-code for the algorithm is presented in Figure 6 for the general case where the query has both spatial (q_R) and temporal (q_T) extents. Purely spatial queries (e.g., find the total sum of measures - throughout history - for regions intersecting q_R) can be answered using only the R-tree, while purely temporal queries (e.g., find the total sum of measures during q_T for all regions) can be answered exclusively by the topmost B-tree. In general, the *aRB-tree* accelerates queries regardless of their selectivity because (i) if the query window q_R (interval q_T) is large, many nodes in the intermediate levels of the R- (B-) tree will be contained in q_R (q_T) so the pre-calculated results are used, and visits to the lower tree levels are avoided; (ii) If q_R (q_T) is small, the *aRB-tree* behaves as a spatio-temporal index.

```

1  function aRB_node_aggregate( $X_i, q_R, q_T$ )
2  //  $X_i$  is a pointer to a node of the aRB-tree. Initially it points to the root
3  //  $q_R$  is the spatial and  $q_T$  the temporal query window, respectively
4      for every entry  $r \in X_i$  do {
5          if ( $q_R$  contains  $r$ .MBR) or ( $X_i$  is a leaf node and  $q_R$  intersects  $r$ .MBR) then {
6              partial_result := B_node_aggregate( $r$ .btree,  $q_T$ ) // visit the corresponding B-tree
7              result :=  $f_{agg}$ (result, partial_result) //  $f_{agg}$  is the aggregation function
8          } else if  $q_R$  partially overlaps  $r$ .MBR then {
9              partial_result := aRB_node_aggregate( $r$ .pointer,  $q_R, q_T$ ) // visit recursively the
10             result :=  $f_{agg}$ (result, partial_result) // aRB sub-tree
11         } // end if
12     } // end for
13     return result

1  function B_node_aggregate( $B_i, q_T$ )
2  //  $B_i$  is a pointer to a node of the B-tree. Initially it points to the root
3      for every entry (i.e. interval)  $b \in B_i$  do {
4          if ( $q_T$  contains  $b$ ) or ( $B_i$  is a leaf node and  $q_T$  intersects  $b$ ) then
5              result :=  $f_{agg}$ (result,  $b$ .aggr) // use the pre-aggregated result
6          else if  $q_T$  partially overlaps  $b$  then {
7              partial_result := B_node_aggregate( $b$ .pointer,  $q_T$ ) // visit recursively the B-subtree
8              result :=  $f_{agg}$ (result, partial_result)
9          } // end if
10     } // end for
11     return result

```

Fig.6. Query processing using the *aRB-tree* (single window queries)

Incremental maintenance of the *aRB-tree* is straightforward. Assume, for example, that at the next timestamp 6 region R_l changes its measure. To update the *aRB-tree*, we first locate R_l in the R-tree (in Figure 5), by performing an ordinary window query using the extent of R_l , after which the B-tree associated with R_l is modified to include the new measure. A change at the lower level may propagate to higher levels; continuing the previous example, after updating R_l .btree, we backtrack to the parent entry R_5 , and modify its B-tree (according to the new aggregate of R_l).

A faster way to perform updates is by following a bottom-up approach². In particular, we can build a hash index on region id and associate each region with a pointer to the last entry of the B-tree that stores its measure. When new information about a region arrives, the hash index is used to locate directly the appropriate B-tree entry where the measure is stored (thus avoiding the window query on the R-tree). Then, the change propagates upwards the B-tree and the R-tree, updating the affected entries. Similar update policies can be applied for volatile regions discussed in subsequent sections.

4.2 The aggregate multi-version R-B-tree

When the extents of data regions change with time, the *aRB-tree* is inadequate because its host index is a spatial access method, which does not support moving objects. To overcome this problem, we propose the *aggregate multi-version R-B-tree* (*aMVRB-tree*), which adopts the *MVR-tree* (discussed in Section 2.1) as the host index. Specifically, each entry r in the *MVR-tree* has the form $\langle r.MBR, r.lifespan, r.aggr, r.pointer, r.btree \rangle$, where (i) the meanings of $r.MBR$, $r.lifespan$, $r.pointer$ are the same as the normal *MVR-tree*, (ii) $r.aggr$ keeps the aggregated measure of r during its lifespan (instead of the whole history as in the *aRB-tree*), and (iii) $r.btree$ points to a B-tree storing its concrete measures. Figure 7 shows an example for the moving regions in Figure 2a. The value 580 stored with R_1 , for example, equals the sum of its aggregate values during interval $[1,4]$ (the lifespan of R_1). On the other hand, the B-tree of R_1' (i.e., the updated version of R_1) contains a single entry (5,130), indicating its measure 130 at the current time 5.

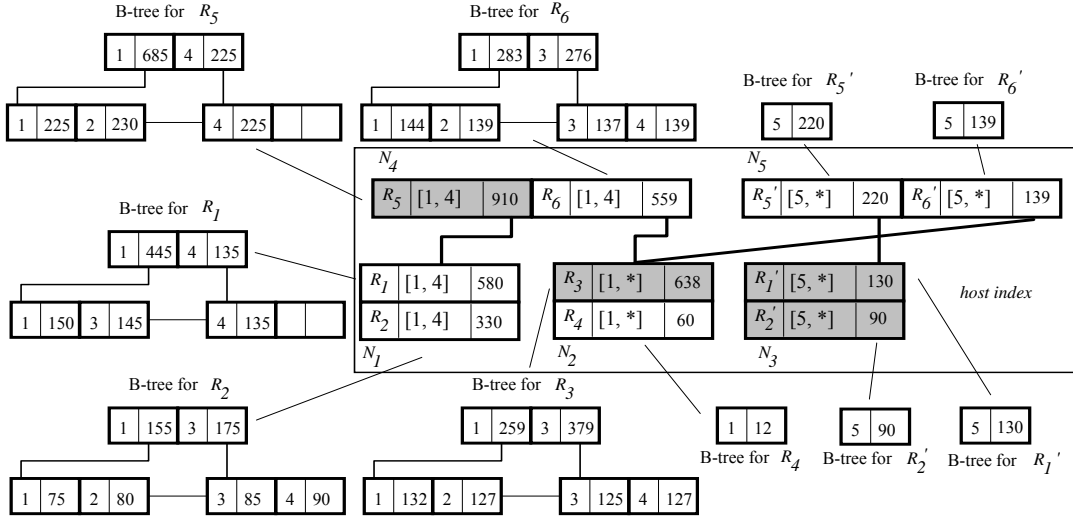


Fig.7. An *aMVRB-tree* (c.f. regions in Figure 2a and measures in Figure 3)

Consider a query asking for the number of phone-calls initiated during interval $q_T=[1,5]$ in all cells intersecting the window q_R in Figure 2a. Since $R_5.MBR$ is inside q_R during time interval $[1,4]$, its child node (at N_4) is not visited. Furthermore, $R_5.btree$ is not retrieved either because its lifespan $[1,4]$ is contained in the query interval $[1,5]$; instead, the summary data (910) of R_5 at node N_4 are simply aggregated. On the other hand, N_2 must be accessed because its parent $R_6.MBR$ partially overlaps q_R . Inside N_2 , only R_3 intersects q_R , and we aggregate its summary

² Bottom up updates using hash indexes have been used extensively in spatio-temporal applications involving

(638) without retrieving its B-tree (as its lifespan $[1,*]=[1,5]$ is also included in q_T). Searching the logical R-tree rooted at N_5 is similar, except that shared nodes should not contribute more than once. Continuing the example, node N_3 is accessed (R_5' partially overlaps q_R) without retrieving any B-tree (because the lifespans of R_1' and R_2' are enclosed by q_T). Further, since N_2 has already been processed, we do not follow $R_6'.pointer$, even though $R_6'.MBR$ partially intersects q_R . In Figure 7, the entries that contribute to the query are shaded.

In order to avoid multiple visits to a shared node via different parents, we search the *MVR-tree* in a breadth-first manner. Specifically, at each level, the algorithm visits all the necessary nodes before descending to the lower level. In Figure 7, for example, nodes N_4 and N_5 (i.e., the root level of the *MVR-tree*) are searched first, after which we obtain an *access list*, containing the ids of nodes N_2, N_3 to be visited at the next level. Thus, multiple visits are trivially avoided by eliminating duplicate entries from the access list. Figure 8 illustrates the complete query algorithm of *aMVRB-trees*, where function *B_node_aggregate* is shown in Figure 6.

```

1  function aMVRB_tree_aggregate( $q_R, q_T$ )
2      initiate an empty access list AL
3      for each root  $r$  responsible for some timestamp of  $q_T$ 
4          let  $r.lifespan$  be the bounding lifespan of all entries in  $r$ 
5          insert ( $r.id, r.lifespan$ ) into AL
6      result := aMVRB_node_aggregate(AL,  $q_R, q_T$ )

1  function aMVRB_node_aggregate(pAL,  $q_R, q_T$ )
2  // pAL (passed from the parent level) is the access list of nodes to be visited at this level
3      result := 0 and initiate an empty access list AL
4      for every entry  $pe \in pAL$  retrieve the node  $X$  whose id equals  $pe.id$ 
6          for every entry  $r \in X$  do {
7              if ( $q_R$  contains  $r.MBR$ ) or ( $X$  is a leaf node and  $q_R$  intersects  $r.MBR$ ) then {
8                  if ( $q_T$  covers the  $r.lifespan$  and  $r.lifespan \subseteq pe.lifespan$ ) then result :=  $f_{agg}(\text{result}, r.agg[])$ 
10                 else { //  $q_T$  intersects but does not cover the lifespan of the entry
11                     partial_result := B_node_aggregation( $r.btree, q_T$ )
12                     result :=  $f_{agg}(\text{result}, \text{partial\_result})$ 
13                 } else if  $q_R$  partially overlaps  $r$  then {
14                     if there is an entry  $e$  in AL whose id equals  $r.pointer$  then
15                          $e.lifespan := e.lifespan \cup (r.lifespan \cap pe.lifespan)$ 
16                     else insert ( $r.pointer, r.lifespan \cap pe.lifespan$ ) into AL
17                 } //end for
18     partial_result := aMVRB_node_aggregate(AL,  $q_R, q_T$ ) //access the next level
19     result :=  $f_{agg}(\text{result}, \text{partial\_result})$ 

```

Fig. 8. Query processing using the *aMVRB-tree*

Note that, the algorithm visits the B-trees of only those entries in the *MVR-tree* whose lifespans cover the starting or ending timestamps of q_T ; for (*MVR*) entries whose lifespans include only the intermediate timestamps of q_T , the relevant aggregate data stored in the *MVR-tree* are used directly. Furthermore, although in Figure 7 we show a separate B-tree for each *MVR-tree* entry, the B-trees of various entries can be stored together in a space efficient manner, described in Section 4.4. Finally, the *aMVRB-tree* can be incrementally maintained in a way similar to *aRB-trees*. Specifically, given the new spatial extent and aggregate value of a region, the update algorithm first locates the corresponding entry in the *MVR-tree* (or inserts an entry if the region incurs extent change), modifies the

intensive updates [Kwon et al. 2002, Lee et al. 2003].

information in its B-tree, and then propagates the changes to higher levels of the *MVR-tree*.

4.3 The aggregate 3-dimensional R-B-tree

As mentioned in Section 2.1, the *MVR-tree* still involves data duplication³, which has negative effects on the space consumption and query performance. To eliminate this problem, we develop the *a3DRB-tree* (aggregate 3-dimensional R-B-tree), by adopting the *3DR-tree* as the host index. Towards this, we follow the “3D box” representation of (discretely) moving rectangles (see Sections 2.1 and 3), but unlike the *a3DR-tree*, a new box is necessary *only* for extent changes (i.e., not for measure changes); hence, there is no redundancy. Specifically, an entry in the host index has the form $\langle r.MBR, r.lifespan, r.btree, r.aggr \rangle$, where $r.MBR$, $r.btree$ are defined as in *aRB-trees*, and $r.aggr$ stores aggregated data over $r.lifespan$. Figure 9 shows an example using the moving regions of Figure 2a. Region R_1 changes to R_1' at timestamp 5, which creates a new box and a new node R_7 containing it.

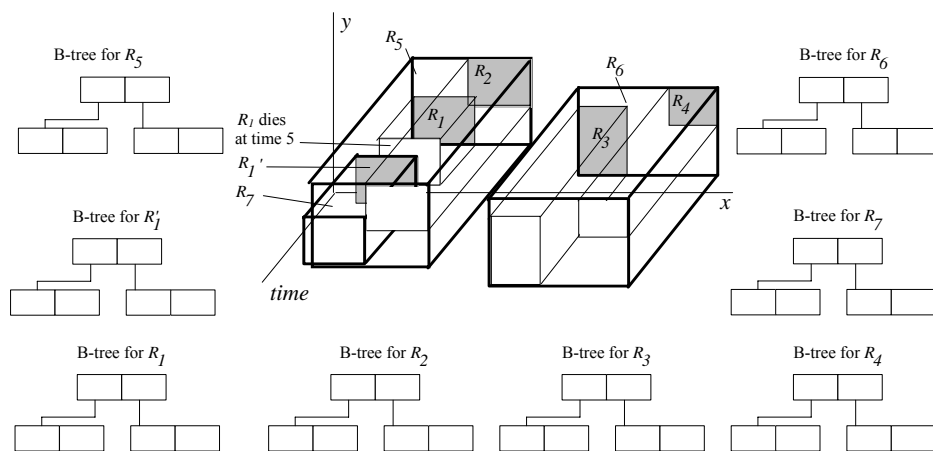


Fig. 9. Example of *a3DRB-tree*

As with the *a3DR-tree*, a spatio-temporal aggregate query is modeled as a 3D box representing the spatial and temporal ranges. The query algorithm follows the same idea as those for *aRB-* and *aMVRB-trees*. Specifically, it starts from the root of the *3DR-tree*, and for each entry r one of the following conditions holds: (i) the entry is covered by both (q_R and q_T) query extents. In this case, its pre-computed aggregate data $r.aggr$ is simply used (subtree or B-tree accesses are avoided), (ii) the entry's spatial extent is covered by q_R , and its temporal extent partially overlaps q_T . The B-tree pointed by $r.btree$ is accessed to retrieve aggregate information for q_T , (iii) the entry's spatial extent partially overlaps q_R , and its temporal extent overlaps (or is inside) q_T . In this case the algorithm descends to the next R-tree level and the same process is applied recursively, and (iv) if none of the previous conditions holds, the entry is ignored.

Although both *aMVRB-* and *a3DRB-trees* aim at volatile regions, they have two important differences. (i) The *a3DRB-tree* maintains a large *3DR-tree* for the whole history, while the *aMVRB-tree* maintains several small trees, each responsible for a relatively short interval. This fact has implications on their query performance as discussed in Section 5. (ii) The *aMVRB-tree* is an *on-line* structure (i.e., it can be incrementally updated), while the *a3DRB-tree*

³ The data duplication in the *MVR-tree* does not involve regions' measures (as is the case in *a3DR-trees*), but is caused by the partially persistent framework.

is *off-line*, meaning that all the region extents must be known in-advance⁴. Specifically, to create an *a3DRB-tree*, we should first build the underlying *3DR-tree* according to regions' spatial extents and lifespans, after which the B-trees of the entries are constructed chronologically by scanning the aggregate changes. Similar to *ARB-* and *AMVRB-trees*, for each aggregate change, the algorithm first identifies the leaf entry of the corresponding region (that produces the change), and then modifies its B-tree. Finally, the update propagates to higher levels of the tree.

4.4 Management of B-trees

Maintaining a separate B-tree for each entry of the *AMVRB-* (*a3DRB-*) tree can lead to considerable waste of space if the B-tree contains too few entries. Consider, for example, Figure 7, where region R_i changes to R_i' at timestamp 5; thus, R_i .btree contains only 4 entries although in practice a page has a capacity of 100-1000 entries. If such situation happens frequently, the average page utilization in the B-trees may be very low. To solve this problem we propose the *B-File (BF)*, which is a space-efficient storage scheme for multiple B-trees. A *BF* possesses the following properties: (i) the B-trees stored in the same *BF* manage disjoint sets of keys, which in our case correspond to timestamps (any timestamp can be indexed by at most one B-tree in the same *BF*), (ii) all the nodes (except, possibly, for the last node of each level) are full (since deletions never happen), and (iii) the search algorithms are the same as those of conventional B-trees (a *BF* is merely a compact storage scheme for multiple B-trees, each maintaining its logical integrity).

Figure 10a illustrates an example *BF*, which stores the B-trees of two regions R and R' (for simplicity, in each B-tree entry we include only the timestamps and not the aggregate values). The lifespan of R is $[1, 19]$, while that of R' is $[20, *]$ (R' is currently alive). The B-tree of R consists of two levels while, up to timestamp 30, the B-tree of R' has only one level. Note that the root pointers of R and R' point to nodes at different levels. The insertion of 35 (in the B-tree of R') causes node B to overflow, and a new node C is created (Figure 10b). An entry 35, pointing to node C , is inserted into A , which becomes the root of the B-tree of R' .

If the live B-tree dies (e.g., R' ceases to exist), the corresponding *BF* becomes *vacant* and may be used for any B-tree created at later timestamps. Whenever a new B-tree needs to be initiated, we first search for vacant *BFs*. If such a *BF* does not exist, a new one is initiated. In practice, the creation of new *BFs* is infrequent because, when an object changes its position or extent, the new entry (in the *MVR-* or *3DR-trees*) can use the vacant *BF* of the previous version. As analyzed in the next section, the *BF* can achieve significant space savings for highly dynamic datasets.

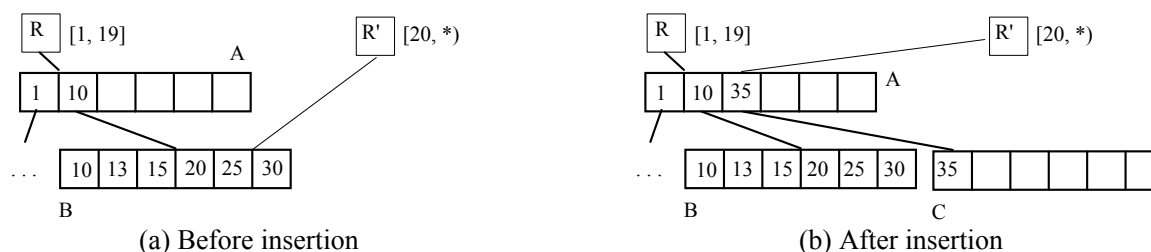


Fig.10. A *B-File* example

⁴ Otherwise, we have to store unbounded boxes inside the *3DR-tree*, which affects query performance severely. The same problem exists for the *a3DR-tree* and, in general, any structure based on *3DR-trees*.

5. PERFORMANCE ANALYSIS

This section theoretically proves the superiority of our solutions and provides cost models for query optimization. Since column scanning ignores the spatial conditions and (as shown in Section 6) has inferior performance, we focus on the *a3DR-tree* and the proposed *aRB-*, *aMVRB-* and *a3DRB-trees* (collectively called *multi-tree structures*). In Section 5.1 we present a unified (high-level) model that describes the behavior of all structures. Then, Sections 5.2-5.4 develop the complete formulae for space consumption and query cost of each method, assuming uniform locations and velocities. Section 5.5 provides significant insight into the characteristics of alternative solutions, and Section 5.6 extends the analysis to general datasets. Table I lists the symbols that will be frequently used in our derivation.

Table I. List of frequent symbols

Symbol	Description
N	total number of data regions
D	region density
T	number of timestamps in history
a_{ext}	extent agility of the dataset
a_{ms}	measure agility of the dataset
q_R, q_S, q_T	query region, side length, and query interval
h	height of the host tree
N_i	number of nodes at the i -th level of the host tree
aP_i	access probability of a level- i node in the host tree
E_i	number of level- i B-trees to be searched
NA_{Bi}	cost of searching a level- i B-tree

5.1 A unified model

To facilitate discussion, let us first consider the following *regular datasets*. At the initial timestamp 1, N regions with density⁵ D distribute uniformly in the 2D unit data space $[0,1]^2$. Then, at each of the subsequent $T-1$ timestamps, (i) a_{ext} percent of the regions change their positions randomly so that the spatial distribution is still uniform (for static dimensions $a_{ext}=0$), and (ii) a_{ms} percent modify their aggregate values, where the *extent* (a_{ext}) and *measure* (a_{ms}) *agilities* remain fixed at all timestamps. Further, each region has the same chance to produce changes, i.e., a_{ext} (a_{ms}) corresponds to the probability that a region changes its extent (measure) at each timestamp. Such regular data allow us to concentrate on the most crucial factors that affect the performance of each method. We will show, in Section 5.6, that the results obtained from the regular case can be easily extended to general datasets (without the above constraints), using histograms.

The objective of analysis is to predict (i) the number of node accesses in answering a spatio-temporal aggregate query, and (ii) the structure size (in terms of the number of nodes). For this purpose, we separate the derivation for the host index (i.e., the *R-*, *MVR-* and *3DR-trees* in the *aRB-*, *aMVRB-* and *a3DRB-trees*, respectively) from that for the measure indexes (i.e., aggregate B-trees). For convenience we say that a measure index (interchangeably, a B-tree) is *at level- i* , if the corresponding host entry (i.e., pointing to the B-tree) is at the i -th level of the host tree. Also,

⁵ The density D of a set of rectangles is the average number of rectangles that contain a given point in space. Equivalently, D can be expressed as the ratio of the sum of the areas of all rectangles over the data space area.

we define the *lifespan of a B-tree node* as the range of timestamps covered by the sub-tree rooted at it. Particularly, the lifespan of the root (of the B-tree) is also the lifespan of the entire B-tree. For example, for R_l .btree (i.e., a level-0 B-tree) in Figure 5, the extents of the first and second leaf nodes are [1,3] and [4,5] respectively, while that of the root is [1,5]. Obviously, the query cost (structure size) equals the sum of the costs (sizes) of the host and measure indexes:

$$NA = NA_{host} + NA_{ms}, \text{ and } Size = Size_{host} + Size_{ms} \quad (5-1)$$

The *a3DR-tree* is a special case of our framework that consists of only the host index, i.e., $Size_{ms}=NA_{ms}=0$ in Equation 5-1. For regular datasets, as defined earlier, the data characteristics are the same across the whole spatio-temporal space, leading to similar properties in all parts of the index. This has several important implications: (i) for all structures, the MBRs of the host entries at the same level have similar sizes, (ii) for *a3DR-*, *aMVRB-*, *a3DRB-trees*, the lengths of the host entries' lifespans are also similar, and (iii) for the proposed structures, the B-trees of the same level manage an equal number of timestamps (i.e., their lifespans are equally long). In particular, property (iii) is most obvious for the *arB-tree*: the B-tree of a host entry at the leaf level indexes all the $a_{ms} \cdot T$ measure changes of the corresponding data region, where a_{ms} and T are the measure agility and number of recorded timestamps, respectively.

Next we investigate Equation 5-1. Let h be the height of the host index (the leaves are at level 0), N_i the number of nodes at the i -th ($0 \leq i \leq h-1$) level, and aP_i the probability that a level- i node is visited for answering a query q . Then, NA_{host} can be represented as:

$$NA_{host} = \sum_{i=0}^{h-1} (N_i \cdot aP_i) \quad (5-2)$$

The above equation already gives the cost (albeit at a coarse level) of the *a3DR-tree* which has no measure indexes. For the proposed multi-tree solutions, we still need to consider NA_{ms} , which depends on two factors: (i) the number E_i of B-trees at the i -th level (of the host index) that need to be searched, and (ii) the cost NA_{Bi} of accessing each level- i B-tree. Then, NA_{ms} (and hence the total cost NA in Equation 5-1) can be derived as: $NA_{ms} = \sum_{i=0}^{h-1} (E_i \cdot NA_{Bi})$, and combining with Equation 5-2,

$$NA = \sum_{i=0}^{h-1} (N_i \cdot aP_i + E_i \cdot NA_{Bi}) \quad (5-3)$$

Now we qualitatively compare, using Equation 5-3, the performance of the *a3DR-tree* and multi-tree structures. Towards this, we relate the query cost to the measure agility a_{ms} that determines the total number of records (recall that $a_{ms} \gg a_{ext}$). In the formula for the *a3DR-tree*, $E_i=NA_{Bi}=0$, but N_i (i.e., the number of nodes at the i -th level) includes all the extent *and* measure changes. In particular, since N_i grows linearly with the measure agility a_{ms} , *the cost of the a3DR-tree is linear to a_{ms}* . On the other hand, for the multi-tree structures, N_i is very low since it is decided by only the number of extent changes (i.e., not related to a_{ms}), which is much smaller than the number of measure changes. As a result, the overall cost NA is dominated by that of searching the measure indexes. Further, as the number E_i of B-trees searched depends only on the host index it is also independent of a_{ms} . As will be explained

shortly, the cost NA_{Bi} of searching each B-tree is logarithmic to the measure agility a_{ms} , and therefore *the overall query time of the multi-tree structures is logarithmic to a_{ms}* , which explains their superiority over the *a3DR-tree*.

NA_{Bi} is logarithmic to a_{ms} because *regardless of how many timestamps are involved in the interval q_T , the query accesses at most two complete paths (from the root to the leaf) in a B-tree*. Recall that a node is accessed, if and only if, its lifespan includes the starting or ending timestamp of q_T , and the number of such nodes at each level (of the B-tree) is at most 2! This is illustrated in Figure 11a, which shows a two-level B-tree and the corresponding query range q_T . Leaf nodes *B* and *D* are visited because their extents partially intersect q_T , while leaf node *C* is not accessed since its extent is contained (in q_T); consequently, the aggregate measure stored in the parent entry *c* is used directly. Figure 11b shows another query, where q_T is not totally contained in the lifespan of the B-tree. In this case, the cost is even lower, i.e., the algorithm only visits a single path from the root to leaf level (e.g., the nodes visited are the root and node *B*).

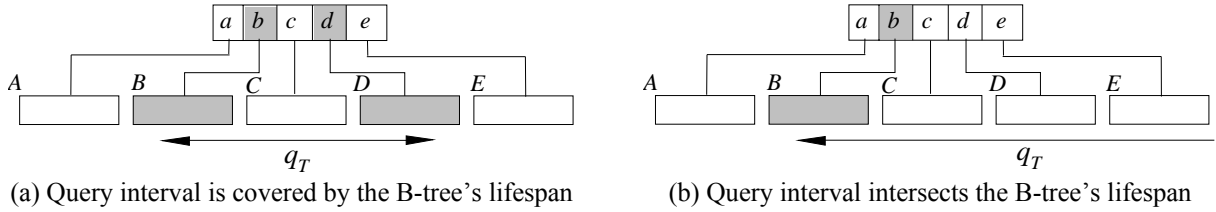


Fig.11. Two cases of searching a measure index

In the *aRB-tree*, a measure index stores all the $a_{ms} \cdot T$ changes of a single (static) data region in history. Hence its height is $\lceil \log_{b_B}(a_{ms} \cdot T / b_B) \rceil$ (where b_B is the node capacity of the B-tree), and NA_{Bi} is at most twice this number. The situation is more complex for the *aMVRB-* and *a3DRB-trees*, but as will be explained in Sections 5.3 and 5.4, the height of a measure index is roughly $\lceil \log_{b_B}[(a_{ms}/a_{ext})/b_B] \rceil$ so that NA_{Bi} is also proportional to $\log(a_{ms})$. In the rest of the section, we extend the above analytical framework for each structure and derive cost models as functions of the data and query properties (specifically, $D, N, T, a_{ms}, a_{ext}, q_R, q_T$). Our discussion utilizes some previous results in the literature of index analysis, which will be well separated from our contributions at the beginning of each subsection.

5.2 Cost model for *aRB-trees*

The analysis of the *aRB-tree* is based on the following lemmas.

Lemma 5.1 [Pagel et al. 1993]: Let r and s be two m -dimensional rectangles that uniformly distribute in the unit universe $[0,1]^m$, and let $r_i (s_i)$ be the side length of $r (s)$ along the i -th dimension ($1 \leq i \leq m$). Then, (i) the probability for r and s to intersect is $\sum_{i=1}^m (r_i + s_i)$, (ii) the probability for r to contain s is $\sum_{i=1}^m (r_i - s_i)$ if $r_i \geq s_i$ for $1 \leq i \leq m$, or 0 otherwise, and (iii) the probability for r to intersect, but not contain, s is $\sum_{i=1}^m (r_i + s_i) - \sum_{i=1}^m (r_i - s_i)$ if $r_i \geq s_i$ on all dimensions $1 \leq i \leq m$, or $\sum_{i=1}^m (r_i + s_i)$ otherwise. ■

Lemma 5.2 [Theodoridis and Sellis 1996]: Let an R-tree indexing N two-dimensional regions with density D that distribute uniformly in the data space. The side length s_i of the MBR of a level- i node ($0 \leq i \leq h-1$, where h is the height of the tree) is:

$$s_i = \sqrt{D_{i+1} \frac{f_R^{i+1}}{N}}$$

where $D_{i+1} = \left(1 + \frac{\sqrt{D_i} - 1}{\sqrt{f_R}}\right)^2$, $D_0 = D$ and f_R is the node fanout of the tree. ■

We first derive the formula that predicts the query cost of the *aRB-tree*, by re-writing the components of Equation 5-3, specifically, h , N_i , aP_i , E_i , NA_{Bi} , as a function of the dataset properties. The first two components are straightforward: given that the R-tree indexes N regions and the node fanout is f_R , the height of the tree $h = \lceil \log_{f_R} (N/f_R) \rceil$, while the number of nodes at the i -th level is $N_i = \lceil N/f_R^{i+1} \rceil$. The derivation of aP_i is also easy. For simplicity, let us consider that the query region q_R is a square⁶ with side length q_S . As discussed in Section 4.1, a node in the R-tree of the *aRB-tree* is searched if and only if its MBR intersects, but is not contained in, q_R . Therefore, according to lemma 5.1 (condition iii), we have (after some simplification) $aP_i = 4 \cdot q_S \cdot s_i$ if $q_S > s_i$, otherwise $aP_i = (q_S + s_i)^2$. Thus it remains to derive E_i (i.e., the number of level- i B-trees searched), and NA_{Bi} (i.e., the number of node accesses in searching a level- i B-tree), for which we prove the following results.

Lemma 5.3: Given an *aRB-tree* and a spatio-temporal aggregation query, whose region is a square with length q_S , the number E_i of B-trees searched at the i -th level of the host index equals:

$$E_0 = \begin{cases} N \left[\left(\sqrt{D/N} + q_S \right)^2 - (q_S - s_0)^2 \right], & \text{if } q_S > s_0 \\ N \left(\sqrt{D/N} + q_S \right)^2, & \text{otherwise} \end{cases} \quad \text{if } i=0 \text{ (leaf level),}$$

$$E_i = \begin{cases} \frac{N}{f_R^i} \left[(q_S - s_{i-1})^2 - (q_S - s_i)^2 \right], & \text{if } q_S > s_{i-1} \text{ and } q_S > s_i \\ \frac{N}{f_R^i} (q_S - s_{i-1})^2, & \text{if } q_S > s_{i-1} \text{ and } q_S < s_i \\ 0, & \text{otherwise} \end{cases} \quad \text{if } i > 0,$$

where the side length s_i of a level- i node in the R-tree is given by Lemma 5.2.

Proof: The B-tree associated with a leaf entry of the R-tree is searched, if and only if (i) the entry's MBR intersects the query region q_R , and (ii) the MBR of the node containing the entry intersects, but is not contained in, q_R . Thus, the number E_0 of such leaf entries equals the difference between the total number of (i) leaf entries (whose MBRs) intersect q_R , and (ii) entries in the leaf nodes completely contained in q_R . Given that there are N (N/f_R) leaf entries (leaf nodes), and the node fanout of the R-tree is f_R , E_0 can be represented as $N \cdot P_1 - f_R \cdot (N/f_R) \cdot P_2$, where P_1 (P_2) denotes the probability that a leaf entry (node) intersects (is contained in) q_R . Since an object (node) MBR is a square with side length $\sqrt{D/N}$ (s_0), the derivation of P_1 and P_2 follows Lemma 5.1 directly, leading to the final representation of E_0 shown in Lemma 5.3.

The derivation of E_i for higher levels $i > 0$ is similar, except that the conditions for a level- i B-tree to be searched is

slightly different. Specifically, the conditions include (i) the corresponding (level- i) host entry's MBR is contained in q_R , and (ii) as with the case of E_0 , the MBR of the node including the entry intersects, but is not contained in, q_R . Given that there are N/f_R^i (N/f_R^{i+1}) entries (nodes) at the i -th level of the R-tree, E_i can be represented as $(N/f_R^i) \cdot P_1' - f_R \cdot (N/f_R^{i+1}) \cdot P_2'$, where P_1' (P_2') is the probability that an entry (node) is contained in q_R . The final form of E_i in Lemma 5.3 is obtained after solving P_1' and P_2' using Lemma 5.1 (applying the MBR extent of the entry/node given in Lemma 5.2). ■

Lemma 5.4: Given an aRB -tree and a spatio-temporal aggregation query, whose interval consists of q_T timestamps, the cost NA_{Bi} of searching a B-tree at the i -th level of the host index equals:

$$NA_{Bi} = \sum_{j=0}^{h_{Bi}-1} NA_{Bij}, \text{ where}$$

$$NA_{Bij} = \begin{cases} \min\left(2, \left\lceil a_{msi} \cdot \frac{T}{b_B^j} \right\rceil\right) & \text{if } q_T > \frac{b_B^j}{a_{msi}} \\ \min\left(1 + q_T \cdot \frac{a_{msi}}{b_B^j}, \left\lceil a_{msi} \cdot \frac{T}{b_B^j} \right\rceil\right) & \text{otherwise} \end{cases}, \quad h_{Bi} = \lceil \log_{b_B}(a_{msi} \cdot T / b_B) \rceil, \quad a_{msi} = 1 - (1 - a_{ms(i-1)})^{f_R} \text{ and } a_{ms0} = a_{ms}$$

Proof: Let us first consider the B-trees associated with the leaf entries of the R-tree. Each of these trees indexes all the measure changes of a particular data region in history, the number of which equals $a_{ms} \cdot T$. Thus, the height of the B-tree equals $h_{Bi} = \lceil \log_{b_B}(a_{ms} \cdot T / b_B) \rceil$. At each level $0 \leq j \leq h_{Bi}-1$ of the B-tree, (i) there are totally $N_{Bj} = \lceil a_{ms} \cdot T / b_B^j \rceil$ nodes, so (ii) each node covers T/N_{Bj} timestamps. As shown in Figure 11a, if the query lifespan q_T is longer than that of a node, two node accesses are necessary (unless level- j is the root). Otherwise, the query only visits those nodes whose lifespans intersect q_T , and according to Lemma 5.1, the probability of such intersection is $(T/N_{Bj} + q_T)/T$. In this case, the expected number NA_{B0j} of node accesses at level- j of the B-tree equals $N_{Bj} \cdot (T/N_{Bj} + q_T)/T = 1 + q_T \cdot a_{ms} / b_B^j$.

The analysis generalizes to the B-trees at higher levels, except that the probability a_{msi} (that a level- i B-tree receives a new measure change at each timestamp) varies. Interestingly, a_{msi} ($i \geq 1$) can be derived from $a_{ms(i-1)}$ based on the following observation (for $i=0$, $a_{ms0} = a_{ms}$). Let e_1 be a level- i entry in the R-tree and e_2 be any entry in the child node of e_1 ; then, whenever a measure change is inserted into the B-tree of e_2 , a change is also inserted into that of e_1 . Given that the average number of entries in the child node of e_1 equals the node fanout f_R , we have $a_{msi} = 1 - (1 - a_{ms(i-1)})^{f_R}$. Replacing a_{ms} with a_{msi} in the derivation for NA_{B0j} , we obtain the same representation for NA_{Bij} , and thus complete the proof. ■

Based on Lemmas 5.3, 5.4, the following theorem presents the query cost (in terms of number of node accesses) of the aRB -tree as a function of the dataset properties and query parameters.

Theorem 5.1 (aRB -tree query cost): Given a spatio-temporal aggregation query, whose region is a square with length q_S and interval includes q_T timestamps, the cost of the aRB -tree equals:

⁶ The simplification of square query windows is common in the R-tree analysis; the extension to general query windows is trivial (according to Lemma 5.1).

$$NA_{aRB} = 4N \cdot q_s \cdot s_0 / f_R + N \left[\left(\sqrt{D/N} + q_s \right)^2 - (q_s - s_0)^2 \right] \left[2 \left\lceil \log_{b_B} \left(a_{ms} \cdot T / b_B \right) \right\rceil - 1 \right]$$

$$\sum_{i=1}^{\lceil \log_{f_R} (N/f_R) \rceil - 1} \left\{ 4N \cdot q_s \cdot s_i / f_R^{i+1} + N / f_R^i \left[(q_s - s_{i-1})^2 - (q_s - s_i)^2 \right] \cdot \left[2 \left\lceil \log_{b_B} \left(a_{msi} \cdot T / b_B \right) \right\rceil - 1 \right] \right\}$$

where N is the dataset cardinality, D the density of data regions, T is the total number of timestamps in history, a_{ms} is the measure agility, f_R the node fanout of the R-tree, b_B is the node capacity of the B-tree, s_i is given in Lemma 5.2, and a_{msi} is given in Lemma 5.4.

Proof: This theorem can be obtained by applying Lemmas 5.3, 5.4 to Equation 5-3. Note that the presented formula corresponds to the costs of “typical” queries, whose regions and intervals are large enough so that we consider the most expensive case in each conditioned expression that appears in Lemmas 5.3 and 5.4. ■

We also prove the following Theorem for the size of the *aRB-tree*.

Theorem 5.2 (*aRB-tree* size): The number of nodes of the *aRB-tree* equals:

$$Size_{aRB} = \sum_{i=0}^{\lceil \log_{f_R} (N/f_R) \rceil - 1} \left[\frac{N}{f_R^{i+1}} + \frac{N}{f_R^i} \sum_{j=0}^{\lceil \log_{b_B} (a_{msi} \cdot T / b_B) \rceil - 1} a_{msi} \cdot \frac{T}{b_B^{j+1}} \right]$$

where a_{msi} is given in Lemma 5.4.

Proof: The number of nodes at the i -th level ($0 \leq i \leq h-1$) of the R-tree is $N_i = N/f_R^{i+1}$, where N is the number of data regions, f_R the node fanout, and $h = \lceil \log_{f_R} (N/f_R) \rceil$ is the height of the tree. Thus, the size of the R-tree is $\sum_{i=0}^{h-1} N/f_R^{i+1}$. As for the measure index size, let us first focus on a level- i B-tree, which, as discussed in Lemma 5.4, indexes $a_{msi} \cdot T$ measures, where a_{msi} is the probability that a new measure is inserted into this tree at each timestamp. Following the same reasoning as the R-tree size analysis, the size of such a B-tree equals $\sum_{j=0}^{h_{B_i}-1} \lceil a_{msi} \cdot T / b_B^{j+1} \rceil$, where $h_{B_i} = \lceil \log_{b_B} (a_{msi} \cdot T / b_B) \rceil$ is the height of the tree, and b_B is the node capacity of the B-tree (recall that each B-tree is packed). Since the total number of level- i B-trees equals N/f_R^i (the entries at the i -th level of the R-tree), the total size of the measure indexes is $\sum_{i=0}^{h-1} (N/f_R^i \cdot \sum_{j=0}^{h_{B_i}-1} \lceil a_{msi} \cdot T / b_B^{j+1} \rceil)$. The formula presented in the theorem corresponds to the sum of the sizes of the host and measure indexes. ■

5.3 Cost model for *aMVRB-trees*

Our analysis of the *aMVRB-tree* uses the following lemma on the *MVR-tree*, which allows us to circumvent the discussion on the complex behavior of multi-version structures.

Lemma 5.5 [Tao et al. 2002a]: Given N regions (with density D) evolving for T timestamps with extent agility a_{ext} , the following estimates hold for the corresponding *MVR-tree*: (i) the height is $h = \lceil \log_{f_{MVR}} (N/f_{MVR}) \rceil$, where f_{MVR} is the node fanout⁷; (ii) the total number of nodes (entries) at the i -th level is $N_i = N/f_{MVR}^{i+1} + a_{ext} \cdot N \cdot (T-1) / (b_{MVR} - f_{MVR})^{i+1}$

⁷ The fanout of the *MVR-tree* should be interpreted as the number of entries in a node that are alive at one timestamp. Note that this is different from the number of entries in a node, which include entries alive at all the timestamps of the node’s lifespan.

$(N_i[b_{MVR} - a_{ext} f_{MVR}^{i+1}/(b_{MVR} - f_{MVR})^i])$, where b_{MVR} is the node capacity; (iii) the side length s_i a node at the i -th level, and the number t_i of timestamps covered by its lifespan, are:

$$s_i = \sqrt{D_{i+1} \frac{f_{MVR}^{i+1}}{N}} \quad D_{i+1} = \left(1 + \frac{\sqrt{D_i - 1}}{\sqrt{f_{MVR}}}\right)^2, D_0 = D \quad t_i = \frac{(b_{MVR} - f_{MVR})^{i+1}}{a_{ext} \cdot f_{MVR}^{i+1}};$$

(iv) the lifespan et_i of an entry at the i -th level covers $t_i f_{MVR}/[b_{MVR} - a_{ext} f_{MVR}^{i+1}/(b_{MVR} - f_{MVR})^i]$ timestamps. ■

The above lemma provides the estimation of h and N_i , while for the other components (i.e., aP_i , E_i , NA_{Bi}) in Equation 5-3, we prove the following results:

Lemma 5.6: Given a spatio-temporal aggregation query, whose region is a square with length q_S and interval includes q_T timestamps, then the probability aP_i that a level- i node of the host *MVR-tree* is accessed, can be represented as: $aP_i = 4 \cdot q_S \cdot s_i (t_i + q_T) / T$, if $q_S > s_i$; otherwise, $aP_i = (q_S + s_i)^2 (t_i + q_T) / T$, where s_i , t_i are given by Lemma 5.5.

Proof: A node in the host *MVR-tree* is searched if (i) its MBR intersects, but is not contained in, the query region q_R , and (ii) its lifespan intersects the query interval q_T . The formulae presented correspond to the product of the probabilities of (i) and (ii) in Lemma 5.1, applying the MBR extent and lifespan of a node/entry in the *MVR-tree* from Lemma 5.5. ■

Lemma 5.7: Given a spatio-temporal aggregation query, whose region is a square with length q_S and interval includes q_T timestamps, the number of level- i B-trees searched in the *aMVRB-tree* equals:

$$E_0 = \begin{cases} N \left[\left(\sqrt{\frac{D}{N}} + q_S \right)^2 - (q_S - s_0)^2 \right] \left(1 + \frac{2q_T - 2}{et_0 + q_T - 1} \right), & \text{if } q_S > s_0 \\ N \left(\sqrt{\frac{D}{N}} + q_S \right)^2 \left(1 + \frac{2q_T - 2}{et_0 + q_T - 1} \right), & \text{otherwise} \end{cases} \quad \text{if } i=0 \text{ (leaf level),}$$

$$E_i = \begin{cases} \frac{N}{f_{MVR}^i} \left[(q_S - s_{i-1})^2 - (q_S - s_i)^2 \right] \left(1 + \frac{2q_T - 2}{et_i + q_T - 1} \right), & \text{if } q_S > s_{i-1} \text{ and } q_S > s_i \\ \frac{N}{f_{MVR}^i} (q_S - s_{i-1})^2 \left(1 + \frac{2q_T - 2}{et_i + q_T - 1} \right), & \text{if } q_S > s_{i-1} \text{ and } q_S < s_i \\ 0, & \text{otherwise} \end{cases} \quad \text{if } i > 0,$$

where s_i , et_i are given in Lemma 5.5.

Proof: A timestamp query is answered in the same way as in the *aRB-tree*, using only the logical R-tree (in the *MVR-tree*) responsible for the query timestamp. Thus, the estimation of E_i is reduced to that of the *aRB-tree* (note that, for $q_T=1$, the presented formulae have the same form as those in Lemma 5.3). For interval queries, since a B-tree is searched only if its host entry's lifespan covers either the starting or ending timestamp of q_T , we compute E_i as $c_1 + c_2 - c_3$, where c_1 (c_2) is the number of B-trees searched at the logical R-tree responsible for starting (ending) timestamp of q_T , and c_3 is the number of *common* B-trees included in c_1 and c_2 (i.e., their host entries cover the entire q_T). Note that c_1 and c_2 are identical because they both correspond to the E_i estimation of timestamp queries, which is already solved earlier (by reducing to the *aRB-tree*). Further, given that the lifespan of a level- i host entry covers

et_i timestamps (given in Lemma 5.5), the probability that the lifespan covers q_T , provided that it covers the starting or ending timestamp of q_T , equals $(et_i - q_T)/(et_i + q_T)$ if $q_T \leq et_i$, in which case c_3 can be obtained as $c_1 \cdot (et_i - q_T)/(et_i + q_T)$. If $q_T > et_i$, then the entry's lifespan cannot contain q_T , and thus $c_3 = 0$. The formulae presented in the lemma correspond to the final form after simplification. ■

Lemma 5.8: Given an *aMVRB-tree* and a spatio-temporal aggregation query, whose query interval consists of q_T timestamps, the cost NA_{Bi} of searching a B-tree at the i -th level of the host index equals:

$$NA_{Bi} = \sum_{j=0}^{h_{Bi}-1} NA_{Bij}, \text{ where:}$$

$$NA_{Bij} = \begin{cases} \min\left(2, \left\lceil \frac{a_{msi} \cdot et_i}{b_B} \right\rceil\right), & \text{if } \min(q_T, et_i) > b_B^j / a_{msi} \\ \min\left(1 + \min(q_T, et_i) \cdot \frac{a_{msi}}{b_B^j}, \left\lceil \frac{a_{msi} \cdot et_i}{b_B} \right\rceil\right), & \text{otherwise} \end{cases}, \quad h_{Bi} = \lceil \log_{b_B}(a_{msi} \cdot et_i / b_B) \rceil, \quad a_{msi} = 1 - (1 - a_{ms(i-1)})^{f_{MVR}},$$

$a_{ms0} = a_{ms}$ and et_i is given in Lemma 5.5.

Proof: This lemma can be proved in the same way as Lemma 5.4, except that each B-tree does not manage all the timestamps in history, but rather the timestamps in the lifespan of the associated host entry. Similar to Lemma 5.4, a_{msi} represents the probability that a new measure change is inserted into the B-tree of a host entry at the i -th level. Consequently, if the lifespan of the entry covers et_i timestamps, its B-tree consists of $a_{msi} \cdot et_i$ records. The correctness follows by replacing $a_{msi} \cdot T$ with $a_{msi} \cdot et_i$ in the proof of Lemma 5.4. ■

Now we are ready to present the complete models for the cost and space consumption of the *aMVRB-tree*.

Theorem 5.3 (aMVRB-tree query cost): Given a spatio-temporal aggregation query, whose region is a square with length q_S and interval includes q_T timestamps, the cost of an *aMVRB-tree* equals:

$$NA_{aMVRB} = 4 \left(\frac{N}{f_{MVR}} + \frac{a_{ext} \cdot N \cdot (T-1)}{b_{MVR} - f_{MVR}} \right) \frac{q_S \cdot s_0 (t_0 + q_T)}{T} + N \left[\left(\sqrt{\frac{D}{N}} + q_S \right)^2 - (q_S - s_0)^2 \right] \left(1 + \frac{2q_T - 2}{et_0 + q_T - 1} \right) (2h_{B0} - 1) +$$

$$\sum_{i=1}^{\lceil \log_{f_{MVR}}(N/f_{MVR}) \rceil - 1} \left\{ 4 \left(\frac{N}{f_{MVR}^{i+1}} + \frac{a_{ext} \cdot N \cdot (T-1)}{(b_{MVR} - f_{MVR})^{i+1}} \right) \frac{q_S \cdot s_i (t_i + q_T)}{T} \right.$$

$$\left. + \frac{N}{f_{MVR}^i} \left[(q_S - s_{i-1})^2 - (q_S - s_i)^2 \right] \left(1 + \frac{2q_T - 2}{et_i + q_T - 1} \right) (2h_{Bi} - 1) \right\}$$

where N is the dataset cardinality, D the density of data regions, T is the total number of timestamps in history, a_{ms} is the measure agility, f_{MVR} the node fanout, b_B is the node capacity of a B-tree, s_i , t_i , et_i are given in Lemma 5.5, and h_{Bi} is given in Lemma 5.8.

Proof: The theorem follows by applying Lemmas 5.6-5.8 to Equation 5-3. ■

Theorem 5.4 (aMVRB-tree size): The number of nodes of the *aMVRB-tree* equals:

$$Size_{aMVRB} = \sum_{i=0}^{\lceil \log_{f_{MVR}}(N/f_{MVR}) \rceil - 1} \left[\frac{N}{f_{MVR}^{i+1}} + \frac{a_{ext} \cdot N \cdot (T-1)}{(b_{MVR} - f_{MVR})^{i+1}} + \frac{N}{f_{MVR}^i} \frac{a_{msi} \cdot T}{b_B} \right]$$

where a_{msi} is given in Lemma 5.8.

Proof: The proof is similar to Theorem 5.2, except that the B-trees of the host entries (in the *MVR-tree*) with disjoint lifespans are stored compactly in a B-File. ■

5.4 Cost models for *a3DR*- and *a3DRB*-trees

The *a3DR-tree* does not involve any measure index, but includes all the extent and measure changes in a single structure. Thus, based on Equation 5-3, its query cost depends only on h , N_i , and aP_i , as solved in the following lemma.

Lemma 5.9: Given N data regions (with density D) evolving for T timestamps with extent (measure) agility a_{ext} (a_{ms}), the total number of records in the *a3DR-tree* equals $N+N\cdot(T-1)\cdot(a_{ext}+a_{ms}-a_{ext}\cdot a_{ms})$. As a result, its height is $h=\log_{f_{3DR}}\{[N+N\cdot(T-1)\cdot(a_{ext}+a_{ms}-a_{ext}\cdot a_{ms})]/f_{3DR}\}$, and the number N_i of nodes at the i -th level equals $N_i=[N+N\cdot(T-1)\cdot(a_{ext}+a_{ms}-a_{ext}\cdot a_{ms})]/f_{3DR}^{i+1}$. The probability aP_i that a level- i node is accessed during a square query, with length q_s and interval q_T , can be represented as:

$$aP_i = \begin{cases} (q_s + s_i)^2 (q_T + t_i)/T - (q_s - s_i)^2 (q_T - t_i)/T, & \text{if } q_s > s_i \text{ and } q_T > t_i \\ (q_s + s_i)^2 (q_T + t_i)/T, & \text{otherwise} \end{cases}$$

where the side length s_i of a node at the i -th level and the number t_i of timestamps covered by its lifespan can be computed using *extent regression functions* [Tao and Papadias 2004]. Note that conventional R-tree analysis (e.g., [Theodoridis and Sellis 1996, Theodoridis et al. 2000]) assumes that nodes have similar extents on each dimension, which does not hold for spatio-temporal applications (nodes may be elongated on the temporal dimension).

Proof: A record is inserted into the *a3DR-tree* at a timestamp when a data region issues an extent or measure change (with probabilities a_{ext} and a_{ms} , respectively). Thus, a region has probability $a_{ext}+a_{ms}-a_{ext}\cdot a_{ms}$ to create a record in the *a3DR-tree* every timestamp. Since the dataset contains N regions evolving for $T-1$ timestamps, the *3DR-tree* has totally $N+N\cdot(T-1)\cdot(a_{ext}+a_{ms}-a_{ext}\cdot a_{ms})$ records. Regarding aP_i , recall that a node in the host *3DR-tree* is searched if its 3D box (bounding its MBR and lifespan) intersects, but is not contained in, the query box (bounding q_R and q_T). The presented equations result from the application of Lemma 5.1. ■

On the other hand, since the host index of the *a3DRB-tree* manages only extent changes, the number of records in it equals $N+N\cdot(T-1)\cdot a_{ext}$; thus, its height is $h=\log_{f_{3DR}}\{[N+N\cdot(T-1)\cdot a_{ext}]/f_{3DR}\}$, and the number N_i of nodes at the i -th level equals $N_i=[N+N\cdot(T-1)\cdot a_{ext}]/f_{3DR}^{i+1}$. Further, since the conditions for a node (in the host index) to be accessed are the same as those for the *aMVRB-tree*, the estimation of aP_i is the same as Lemma 5.6. Similarly, Lemmas 5.7 and 5.8 also predict E_i and NA_{Bi} for the *a3DRB-tree*, except that (i) f_{MVR} should be replaced as $f_{3DR}\cdot t_{i-1}/t_i$ (for $i\geq 1$), and (ii) $et_0=1/a_{ext}$ (where a_{ext} is the extent agility), while $et_i=t_{i-1}$ for $i\geq 1$. The following theorems summarize the complete models for the *a3DR*- and *a3DRB*-trees. The sizes of the *a3DR-tree* and *a3DRB-trees* are derived in a way similar to Theorems 5.2 and 5.4.

Theorem 5.5 (*a3DR*-, *a3DRB*-trees query costs): Given a spatio-temporal aggregation query, whose region is a square with length q_s and interval includes q_T timestamps, the costs of the *a3DR*- and *a3DRB*-trees are:

$$\begin{aligned}
NA_{a3DR} &= \sum_{i=0}^{h-1} \frac{N + N(T-1)(a_{ext} + a_{ms} - a_{ext} \cdot a_{ms})}{f_{3DR}^{i+1} \cdot T} \left[(q_s + s_i)^2 (q_T + t_i) - (q_s - s_i)^2 (q_T - t_i) \right] \\
NA_{a3DRB} &= 4 \left(\frac{N + a_{ext} \cdot N \cdot (T-1)}{f_{3DR}} \right) \frac{q_s \cdot s_0 (t_0 + q_T)}{T} + N \left[\left(\sqrt{D/N} + q_s \right)^2 - (q_s - s_0)^2 \right] \left(1 + \frac{2q_T - 2}{1/a_{ext} + q_T - 1} \right) (2h_{B_0} - 1) + \\
&\quad \left. \sum_{i=1}^{\lceil \log_{f_{3DR}}(N/f_{3DR}) \rceil - 1} \left\{ 4 \left(\frac{N + a_{ext} \cdot N \cdot (T-1)}{f_{3DR}^{i+1}} \right) \frac{q_s \cdot s_i (t_i + q_T)}{T} \right. \right. \\
&\quad \left. \left. + N / \left(f_{3DR} \cdot t_{i-1} / t_i \right)^i \left[(q_s - s_{i-1})^2 - (q_s - s_i)^2 \right] \left(1 + \frac{2q_T - 2}{s_{i-1} + q_T - 1} \right) (2h_{B_i} - 1) \right\} \right.
\end{aligned}$$

where N is the dataset cardinality, D the density of, T the total number of timestamps in history, a_{ext} (a_{ms}) the extent (measure) agility, f_{3DR} the node fanout of the $3DR$ -tree (its value is different in the $a3DR$ - and $a3DRB$ -tree), s_i , t_i , et_i are computed using extent regression functions, and h_{B_i} , a_{msi} are given in Lemma 5.8.

Proof: The theorem follows by applying Lemmas 5.6-5.9 to Equation 5-3. ■

Theorem 5.6 ($a3DR$ -, $a3DRB$ -tree sizes): The number of nodes of the $a3DR$ -, and $a3DRB$ -trees is:

$$\begin{aligned}
Size_{a3DR} &= \sum_{i=0}^{h-1} \frac{N + N(T-1)(a_{ext} + a_{ms} - a_{ext} \cdot a_{ms})}{f_{3DR}^{i+1}} \\
Size_{a3DRB} &= \sum_{i=0}^{\lceil \log_{f_{3DR}}(N/f_{3DR}) \rceil - 1} \left[\left(\frac{N + a_{ext} \cdot N \cdot (T-1)}{f_{3DR}^{i+1}} \right) + N / f_{3DR}^i \frac{a_{msi} \cdot T}{b_B} \right]
\end{aligned}$$

where N is the dataset cardinality, D the density of data regions, T is the total number of timestamps in history, a_{ext} (a_{ms}) is the extent (measure) agility, b_B is the node capacity of a B-tree and f_{3DR} the node fanout of the $3DR$ -tree, and a_{msi} is given in Lemma 5.8.

Proof: By lemma 5.9, the total number of entries in the $a3DR$ -tree is $N+N \cdot (T-1) \cdot (a_{ext} + a_{ms} - a_{ext} \cdot a_{ms})$, after which the structure size can be obtained in the same way as the R-tree size estimation [Theodoridis and Sellis 1996, Tao and Papadias 2004]. The proof for the size of the $a3DRB$ -tree is similar to that of Theorem 5.4. ■

5.5 Performance characteristics and simplified models

The previous equations can be used directly for query optimization. Furthermore, they mathematically reveal the factors that determine the performance of each structure and promote our understanding about their behavior. The first observation, which leads to simplification of the formulae, is that the query cost of each method involves a dominant term. Specifically, the cost of the $a3DR$ -tree is dominated by the number of leaf node accesses (a typical phenomenon for multi-dimensional indexes). For the proposed multi-tree structures, however, the cost is dominated by that of searching the B-trees associated with the leaf entries in the host index. In other words, the query cost on the host index is negligible compared to the total processing time, because a leaf node access in the host index usually necessitates visits to the associated B-trees (each involving at least one node access). Based on this fact, we can simplify the cost of the aRB -tree (Theorem 5.1) into:

$$NA_{aRB} = 2N \left[\left(\sqrt{\frac{D}{N}} + q_s \right)^2 - \left(q_s - \sqrt{\frac{f_R}{N}} \right)^2 \right] \left\lceil \log_{b_B} \left(\frac{a_{ms} \cdot T}{b_B} \right) \right\rceil \quad (5-4)$$

Its advantage over the *a3DR-tree* (given in Theorem 5.5, setting a_{ext} to 0) becomes obvious: its cost increases only logarithmically with a_{ms} , while that of the *a3DR-tree* (Theorem 5.5) grows linearly. This and the subsequent observations are experimentally confirmed in Section 6.

Regarding the solutions for volatile data regions, an important fact is that for *aMVRB-* and *a3DRB-trees* the height of each B-tree associated with a leaf entry in the host index is usually 1 in practice, indicating that the total number of node accesses equals the number of qualifying B-trees. In particular, for the *a3DRB-tree*, the height $h_{B0} = \lceil \log_{b_B} [(a_{ms}/a_{ext})/b_B] \rceil$ (Lemma 5.8) equals 1 as long as $a_{ms}/a_{ext} < b_B$. Given that for typical page sizes, b_B is 100-1000, this condition holds when the regions' measures change less than 100-1000 times faster than their extents. For the *aMVRB-tree*, on the other hand, $h_{B0} = \lceil \log_{b_B} [(a_{ms} \cdot et_0)/b_B] \rceil$ (Lemma 5.8), where et_0 (given in Lemma 5.5) equals $(b_{MVR} - f_{MVR})/[a_{ext}(b_{MVR} - a_{ext} \cdot f_{MVR})] \leq 1/a_{ext}$ (recall that $a_{ext} \leq 1$). Thus, for *aMVRB-trees*, $h_{B0} \leq \lceil \log_{b_B} [(a_{ms}/a_{ext})/b_B] \rceil$, i.e., the condition for $h_{B0}=1$ is even easier to satisfy than *a3DRB-trees*. When the height of measure indexes equals 1, the query cost of these two structures can be simplified as follows:

$$NA_{a3DRB} = N \left[\left(\sqrt{\frac{D}{N}} + q_s \right)^2 - (q_s - s_0)^2 \right] \left(1 + \frac{2q_T - 2}{1/a_{ext} + q_T - 1} \right) \quad (5-5)$$

$$NA_{aMVRB} = N \left[\left(\sqrt{\frac{D}{N}} + q_s \right)^2 - (q_s - s_0)^2 \right] \left(1 + \frac{2q_T - 2}{\frac{1}{a_{ext}} \frac{b_{MVR} - f_{MVR}}{b_{MVR} - a_{ext} \cdot f_{MVR}} + q_T - 1} \right) \quad (5-6)$$

In this case the query time of the *aMVRB-* and *a3DRB-trees* is independent of the measure agility a_{ms} , in contrast to the linear deterioration of the *a3DR-tree* (Theorem 5.5). Further, the relative performance of the two multi-tree structures is also clear (through the comparison⁸ of Equations 5-5 and 5-6): the *a3DRB-tree* always outperforms the *aMVRB-tree* (i.e., $(b_{MVR} - f_{MVR})/[a_{ext}(b_{MVR} - a_{ext} \cdot f_{MVR})] \leq 1/a_{ext}$) except for $q_T=1$ (i.e., timestamp queries). Nevertheless, recall that the *aMVRB-tree* has wider applicability since it is an on-line structure, while the *a3DRB-tree* does not support incremental updates.

Finally, the size comparison of the *a3DR-tree* and the proposed methods is obvious: the multi-trees consume significantly less space due to the lack of redundancy, which is also reflected in the cost models (Theorems 5.2, 5.4, 5.6). Specifically, observe that the total number of records stored in all trees is approximately the same; however, the *a3DR-tree* has rather low fanout f_{a3DR} (since each entry of the tree must store both extent and measure information), while for multi-tree structures, most data (i.e., the measures) are stored in packed B-trees with large node capacity b_B ($\approx 3f_{a3DR}$ in our experimental settings), hence requiring fewer nodes.

⁸ Strictly speaking, the two equations are not directly comparable due to the different estimates of s_0 (i.e., the side length of the MBR of a leaf node). Nevertheless, the difference is small enough for our discussion to hold.

5.6 Extension to general datasets

So far we have focused on regular datasets, where the spatial distribution remains uniform and the aggregate and extent agilities are constant throughout the history. As discussed in [Tao et al. 2002a], the analysis on general datasets (e.g., non-uniform spatial distribution, variable agilities at different timestamps, etc.) can be reduced to that of regular data, based on the fact that even though the overall data distribution may deviate significantly, for typical queries with small regions (compared to the data space) and intervals (compared to the history length), the distribution of data satisfying the query conditions is usually fairly regular. This permits the application of the regular model at the query spatial and temporal extents, after the local data properties (i.e., data density, average agilities, etc) are accurately estimated, which can be achieved through histograms [Tao et al. 2002a]. We adopt the same approach in our experimental evaluation for providing cost estimations to general data. Finally, note that all the proposed equations for structure sizes directly support non-regular data (i.e., without the need of histograms).

6. EXPERIMENTS

In this section, we evaluate the proposed methods under a variety of experimental settings. The spatial datasets used in the following experiments include [Tiger]: (i) *LA* that contains 130k rectangles representing street locations, and (ii) *LB* that consists of 50k road segments. Due to the lack of real spatio-temporal (aggregation) data, datasets with static regions are created as follows. At timestamp 0, each object (in a unit spatial universe) of a real dataset is associated with a measure (uniformly distributed in $[0,10000]$). Then, for each of the subsequent 999 timestamps (i.e., $T=1000$), a_{ms} percent of the objects are randomly selected to change their measures by offsets uniformly decided in $[-100,100]$. Dynamic datasets (volatile regions) are synthesized in a similar manner except that at each timestamp, a_{ext} percent of the regions move their centroids (towards random directions) by distances that are uniform in $[-0.01, 0.01]$. We vary a_{ms} as a dataset parameter from 1% to 20%, and a_{ext} from 1% to 9%, resulting in a total of 1 to 20 million records. In most of the combinations of a_{ms} and a_{ext} , the measure agility is (up to 20 times) larger than the extent agility. Figure 12 shows the visualization of *LA* and *LB*, while Figure 13 illustrates the distributions of dynamic data (created from *LA* with $a_{ext}=5\%$) at timestamps 250, 500, and 1000. Notice that the distribution gradually becomes uniform.

All implementations of R-trees use the R*-tree [Beckmann et al. 1990] update algorithms. The node size is set to 1k bytes, so that the node capacity of the R-tree (*MVR-tree*) is 36 (28) entries. The *3DR-trees* used in *a3DR-*, and *a3DRB-trees* have slightly different entry formats, resulting in capacities 31 and 28, respectively. The node capacity of a B-tree equals 127 in all cases. Each query specifies a square spatial region with side length q_S (i.e., if $q_S=0.1$, the query occupies 1% area of the universe), and a temporal interval involving q_T timestamps. The distribution of the query regions follows that of data in order to avoid queries with empty results, while the temporal interval is generated uniformly in $[1,1000]$ (i.e., the entire history). The cost of a structure is measured as the average number of node accesses for answering a workload of 200 queries with the same parameters q_S and q_T . In the next section, we first measure the performance (i.e., size and query costs) of alternative methods, and then evaluate the accuracy of the proposed cost models in Section 6.2.

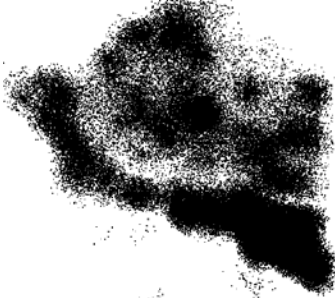


LA

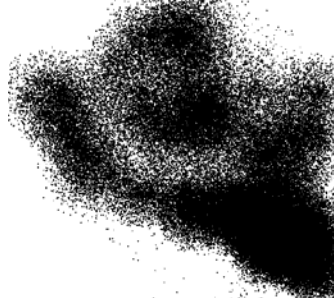


LB

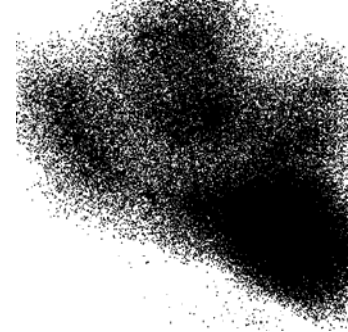
Fig.12. Spatial distributions



Timestamp 250



Timestamp 500



Timestamp 1000

Fig.13. Distribution changes of dynamic data (created from LA)

6.1 Structure size and query performance

We start from static regions (i.e., $a_{ext}=0$), and compare the proposed *aRB-tree* with existing solutions (described in Section 3), namely, column scanning (*ColS* for short), and the *a3DR-tree*. The first experiment evaluates space consumption by varying the measure agility a_{ms} from 1% to 20%. As shown in Figure 14, the *aRB-tree* is the smallest structure in all cases. Despite the intermediate tree levels, it consumes less space than *ColS*, because it does not replicate (in the B-trees) measures that remain constant. On the other hand the fact table approach has to create a new column for each timestamp. The *aRB-tree* size is constant until 10% agility, after which it stabilizes at some higher values. This is because, when the agility exceeds 10%, the height of a data region's B-tree increases by one level. Notice that for $a_{ms}=10\%$, each data region issues around 100 ($=a_{ms} \cdot T$) aggregate updates throughout the history, which is smaller than the B-tree node capacity (127), i.e., each B-tree has one node. Similarly, for $a_{ms}=15\%$ each B-tree manages on the average 150 records, thus it requires two levels. The *a3DR-tree*, on the other hand, grows linearly with a_{ms} and consumes more space than *ColS* for $a_{ms}>10\%$.

The next experiment measures the query cost as a function of q_s (i.e., the extent of the query MBR), by fixing q_t to 100 timestamps (i.e., 10% of the history) and a_{ms} to 10%. Figure 15a shows the results for dataset *LA*, varying q_s from 0.1 to 0.5. The *aRB-tree* outperforms its competitors significantly for all q_s (notice the logarithmic scale). Furthermore, the costs of the *aRB-* and *a3DR-trees* initially increase with q_s , but decrease after q_s exceeds 0.4. This is not surprising because, for skewed distributions (see Figure 12), a large query will contain the MBRs of most nodes, thus resulting in fewer node accesses. Similar phenomena have also been observed in [Tao et al. 2002a] for spatial aggregation. *ColS* is worse than the other methods by more than an order of magnitude, because it retrieves

the information of all regions at each queried timestamp, and hence its cost is linear to q_T , but not affected by q_S . Since *ColS* is significantly more expensive (by orders of magnitude) in all our experiments, we omit its results in the sequel.

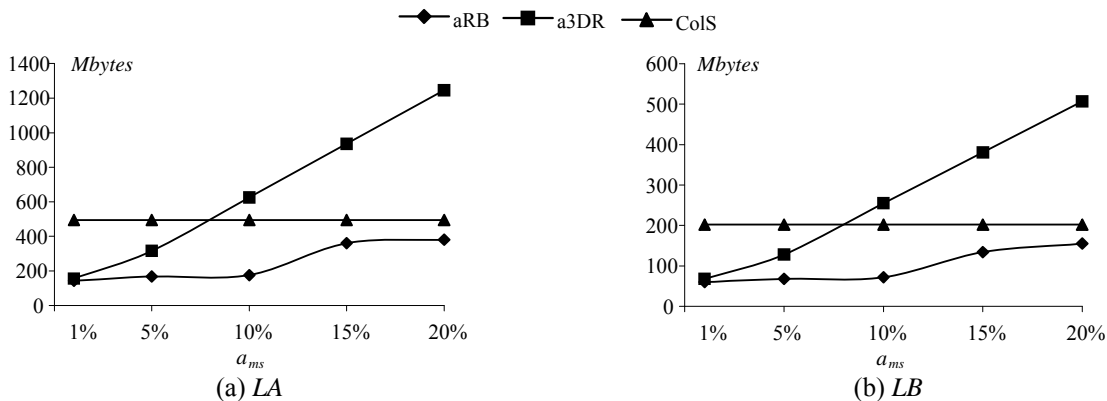


Fig.14. Size vs. measure agility

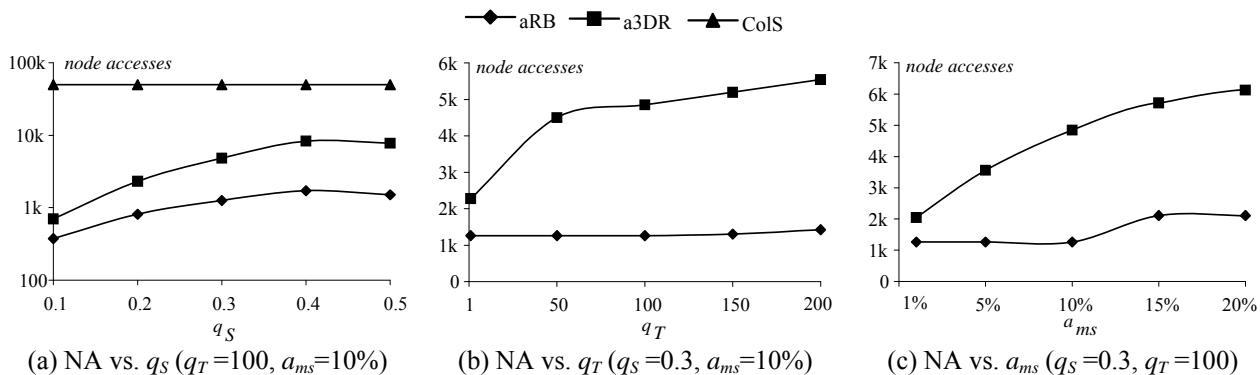


Fig.15. Node accesses for LA (static regions)

Next we fix q_S to 0.3, a_{ms} to 10%, and increase q_T from 1 to 200 timestamps. Figure 15b illustrates the number of node accesses as a function of q_T for the *aRB*- and *a3DR*-trees. The performance of the *aRB*-tree does not deteriorate with q_T because as discussed in Section 5, the cost is dominated by q_S (which determines the number of host entries whose B-trees need to be searched), while visiting each B-tree has almost the same overhead. The *a3DR*-tree, however, deteriorates very fast with q_T , and becomes almost five times slower than the *aRB*-tree when $q_T=200$. In Figure 15c, q_S and q_T are set to their median values 0.3 and 100 respectively, and a_{ms} ranges from 1% to 20%. The cost of the *aRB*-tree remains the same until $a_{ms}=10\%$ because as discussed for Figure 14, the B-tree height of each host entry does not change until this agility. For $a_{ms}\geq 15\%$, each B-tree contains one more level which almost doubles the query cost. It is worth mentioning that the *aRB*-tree will not deteriorate until the B-tree height increases again, which however, will happen only at much higher agility, due to the fact that the height grows logarithmically with the cardinality. Figure 16 shows the results of the same experiments for dataset *LB*, where similar phenomena can be observed. In summary, the *aRB*-tree is clearly the most efficient structure for static regions, while at the same time it consumes less space than the other approaches.

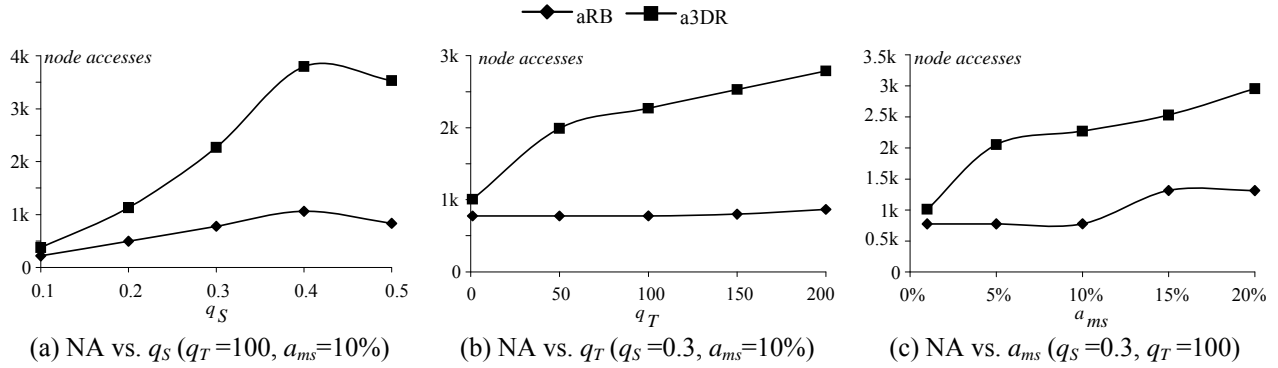


Fig.16. Node accesses for *LB* (static regions)

Having presented the results for static regions, we now proceed with volatile data (where a dataset is described by both the aggregate a_{ms} and extent a_{ext} agilities), and compare the $aMVRB$ - and $a3DRB$ -trees against the $a3DR$ -tree. Figure 17a (17b) plots the index sizes for dataset *LA* as a function of a_{ms} (a_{ext}), by fixing a_{ext} (a_{ms}) to 5% (10%). Observe that $a3DRB$ -trees are the smallest in all cases because they do not incur redundancy. The $aMVRB$ -tree consumes less space than the $a3DR$ -tree unless $a_{ms} \leq 5\%$ ($a_{ext} \geq 7\%$) in Figure 17a (17b), because for small a_{ms} (large a_{ext}), there are relatively few measure (many extent) changes; thus the size of an $aMVRB$ -tree is dominated by the MVR -tree which, due to the data duplication introduced by the multi-version technique, is larger than the $a3DR$ -tree. Figure 18 shows similar results for dataset *LB*.

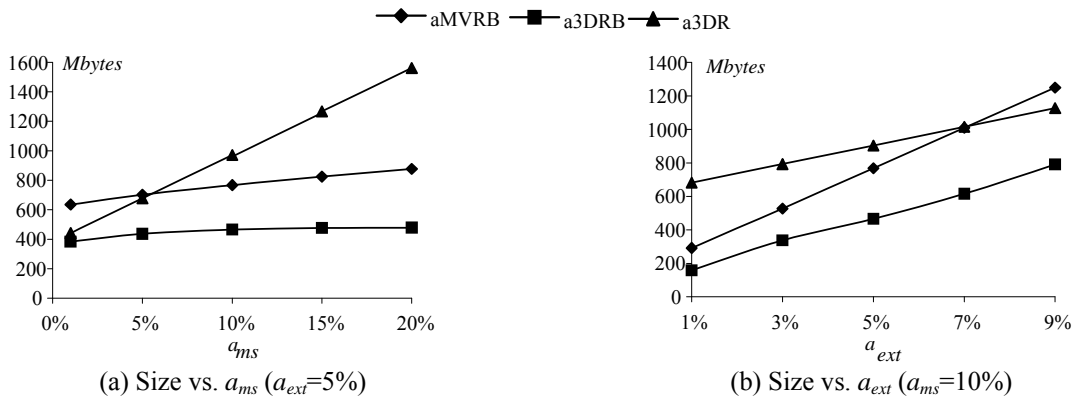


Fig.17. Structure sizes for *LA* (volatile regions)

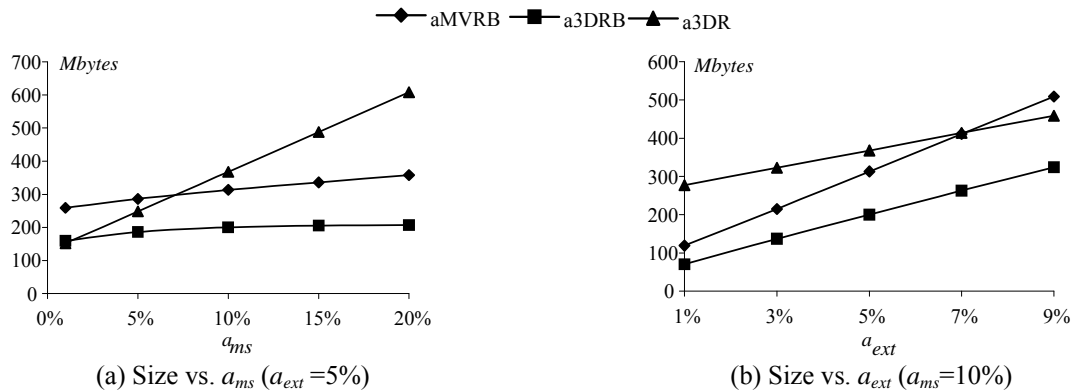


Fig.18. Structure sizes for *LB* (volatile regions)

The previous diagrams (Figures 17-18) for size evaluation of *aMVRB*- and *a3DRB*-trees correspond to an implementation using B-Files. Figures 19a and 19b illustrate the benefit ratio, i.e., the ratio of space without/with B-Files, as a function of a_{ms} and a_{ext} , respectively for *LA* (the results for *LB* are similar). The inclusion of B-Files results in structures that are between 10 and 27 times smaller. The *aMVRB*-tree receives higher improvements than the corresponding *a3DRB*-tree, due to the fact that it contains more host entries and thus requires a larger number of B-trees, leading to more space waste if B-Files are not used. For all subsequent experiments we employ the B-File implementation.

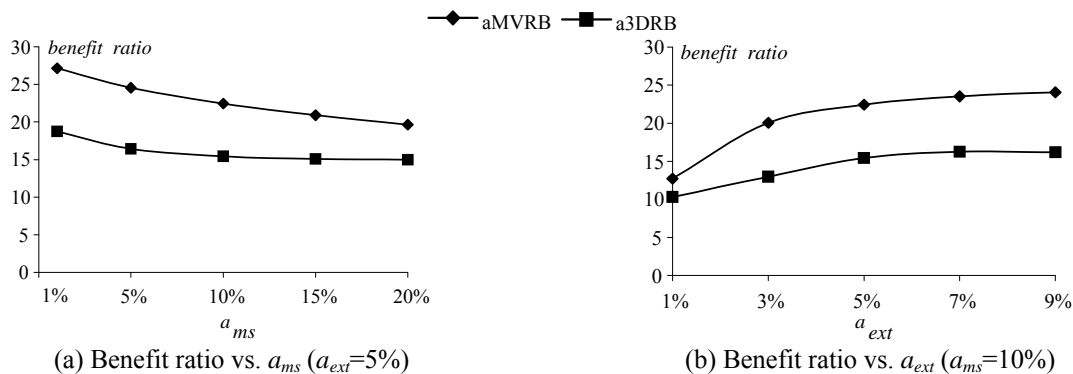


Fig.19. Benefits of using B-Files for *LA* (volatile regions)

The next set of experiments evaluates the query performance of methods for volatile regions. In Figure 20a, we fix q_T , a_{ms} , and a_{ext} to their median values, and measure the query cost as a function of q_S . As expected, the proposed structures outperform the *a3DR*-tree significantly, while the *a3DRB*-tree is even more efficient than the *aMVRB*-tree.

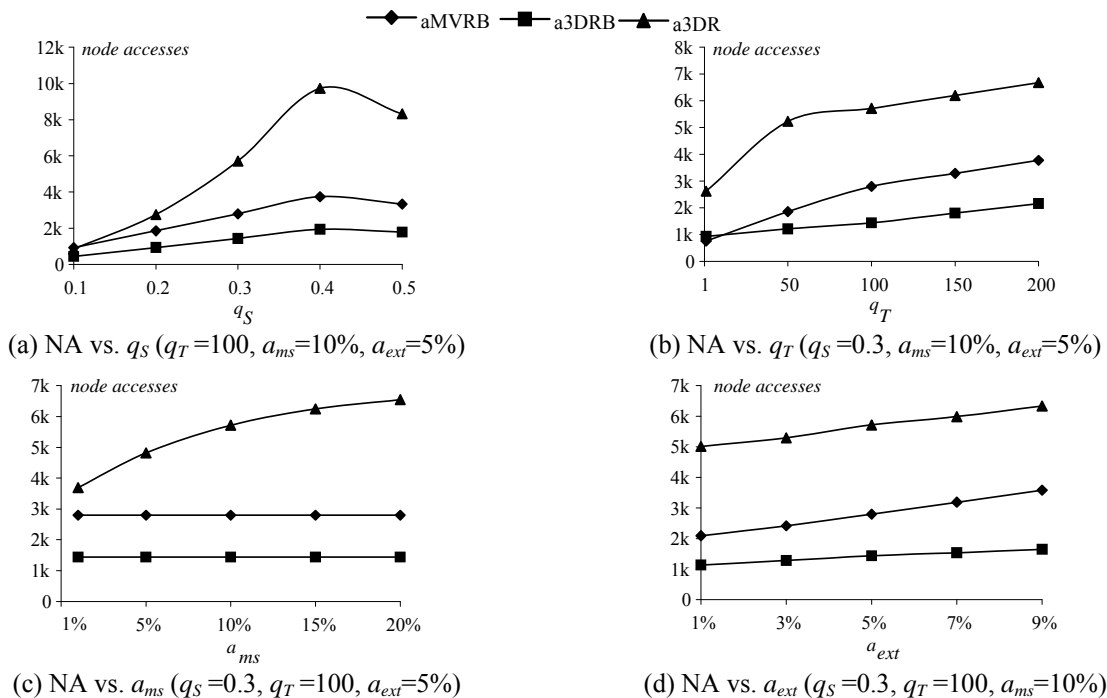


Fig.20. Node accesses for *LA* (volatile regions)

Similar to Figure 15a, the costs of all approaches initially grow with q_s , but decrease after the query becomes sufficiently large ($q_s > 0.4$). Figure 20b shows the number of node accesses as a function of q_T , fixing q_s , a_{ms} , a_{ext} to 0.3, 10%, 5% respectively. As predicted in Section 5.5, the *aMVRB-tree* performs better than the *a3DRB-tree* for timestamp queries (i.e., $q_T=1$), for which only one logical R-tree (in the *aMVRB-tree*) is visited. The *a3DRB-tree* is the best structure for the other values of q_T , while the *a3DR-tree* yields the worst performance in all cases. Figure 20c shows the cost by varying a_{ms} from 1% to 20%. Although the performance of *a3DR-tree* deteriorates significantly when a_{ms} increases, the costs of *aMVRB-* and *a3DRB-*tees are not affected at all. Figure 20d demonstrates the node accesses by varying a_{ext} . In general, the *a3DRB-tree* has the best performance (and the smallest size), followed by the *aMVRB-tree*. However, the *aMVRB-tree* is the only on-line structure, applicable in cases where the region extents are not known in advance. The results for dataset *LB* are similar and omitted.

6.2 Accuracy of the cost models

This section evaluates the accuracy of the cost models proposed in Section 5. Given the actual *act* and estimated *est* values, the relative error is defined as $|act-est|/act$. Based on this, we measure the error for a query workload as the average error of all queries involved. In order to estimate the performance for non-regular data distributions, we maintain histograms as described in [Theodoridis et al. 2000, Tao et al. 2002a]. Specifically, the histogram for static regions consists of a grid with $H \times H$ cells that partition the space regularly, and each cell is associated with its local density⁹. For volatile data, the histogram contains a set of grids such that the i -th grid corresponds to the data distribution at the $100 \cdot i$ -th timestamp (i.e., for $T=1000$, 11 grids are maintained). Since the variation of distribution is slow with time, the i -th grid can be used to represent the distributions between the $100 \cdot i$ -th and $(100 \cdot i + 99)$ -th timestamps [Tao et al. 2002a].

Starting with static regions, Figure 21 shows the relative error (as a function of a_{ms}) of Theorem 5.2 (5.6) that computes the size of the *aRB-tree* (*a3DR-tree*) for datasets *LA* and *LB*. The estimated values are very accurate (maximum error 3%) and the precision increases with a_{ms} . The minimum error will be achieved when $a_{ms}=100\%$, in which case the size estimation of the *aRB-tree* becomes trivial because each B-tree of the host entry simply contains exactly T records, where T is the number of timestamps in history.

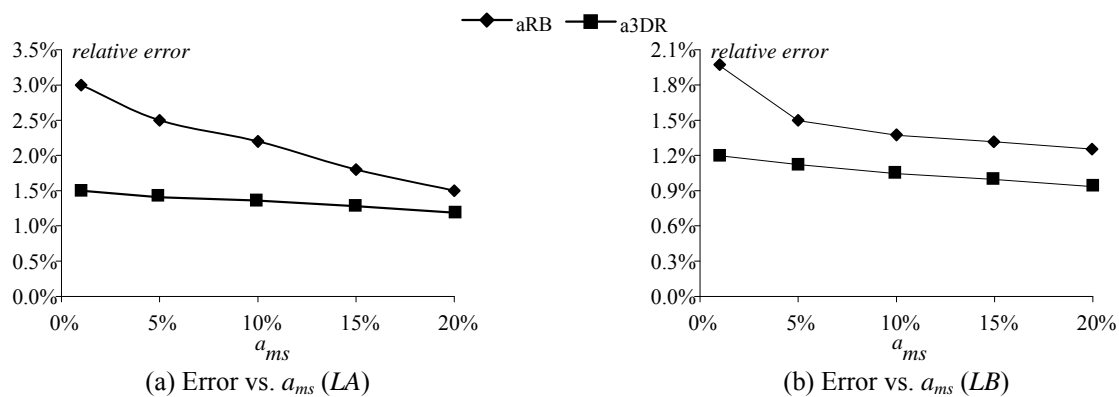


Fig. 21. Size estimation accuracy

Next we evaluate Theorems 5.1, 5.5 that predict the number of node accesses for *aRB*- and *a3DR*-trees. Figures 22a, 22b, 22c illustrate the error as functions of q_S , q_T , q_A for dataset *LA* (by setting the other parameters to their median values in each case). A general observation is that queries incurring higher overhead can usually be better predicted, which is consistent with previous spatial analysis [Theodoridis and Sellis 1996, Theodoridis et al. 2000, Acharya et al. 1999]. In Figure 22a, for example, the error initially drops with q_S but grows after q_S exceeds 0.4, corresponding to the same behavior as Figure 15a. Similar phenomena can also be observed in Figures 22b and 22c, where the settings are the same as Figures 15b and 15c, respectively, as well as Figure 23 for dataset *LB*. It is worth mentioning that the maximum error (about 20%) in query cost estimation is higher than that of size estimation (Figure 21), because, as indicated in Theorems 5.1 and 5.5, the cost depends, not only on the structure size, but also on the node extents. Hence the overall error accumulates that of estimating the node extents (i.e., the imprecision of the previous cost models such as the one in Lemma 5.2), and the inaccuracy introduced by the histogram.

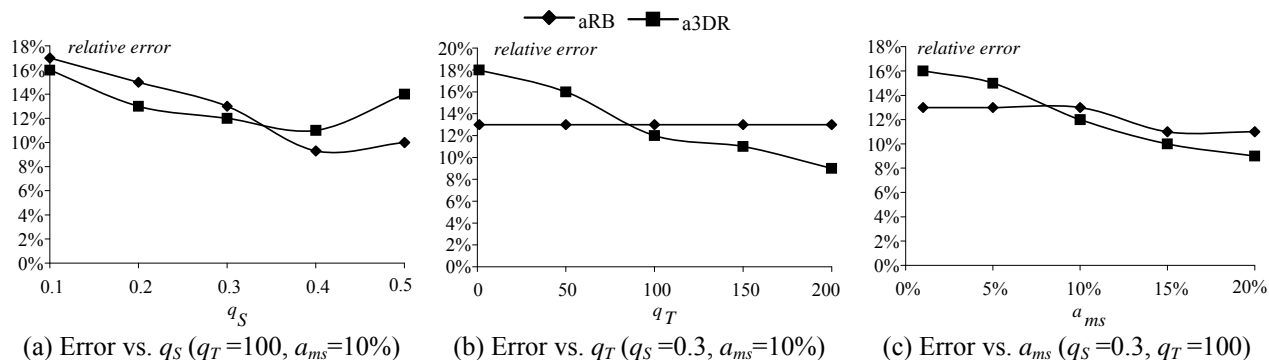


Fig.22. Node access estimation accuracy for *LA* (static regions)

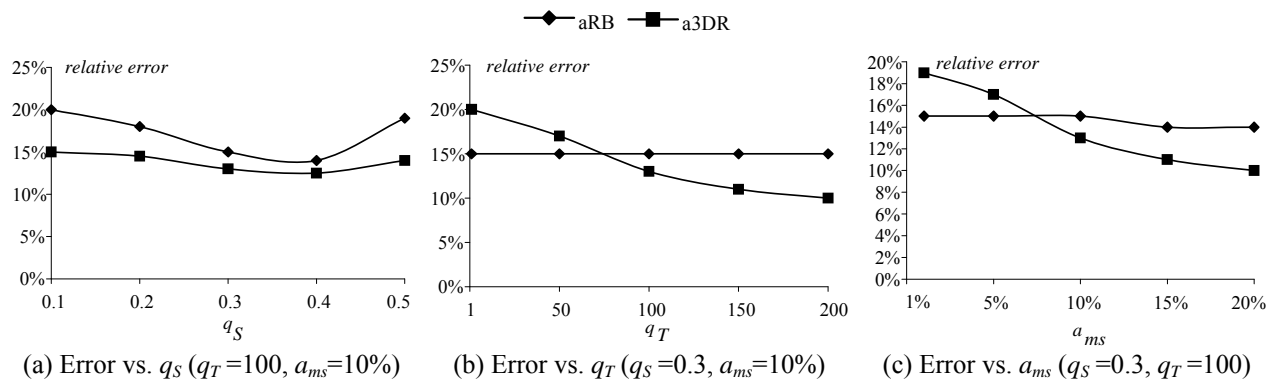


Fig.23. Node access estimation accuracy for *LB* (static regions)

The last set of experiments evaluates the accuracy of the cost models for volatile regions. Figure 24 shows the error of estimating the sizes of the proposed structures on both datasets *LA* and *LB*. The estimated values are accurate (maximum error 7%) and the precision improves with extent and aggregate agilities.

⁹ Assuming there are n data rectangles r_i ($1 \leq i \leq n$) intersecting a cell c , the local density of c is defined as $\sum x_i / \text{area}(c)$ where x_i is the intersection area between c and r_i , and $\text{area}(c)$ is the area of c .

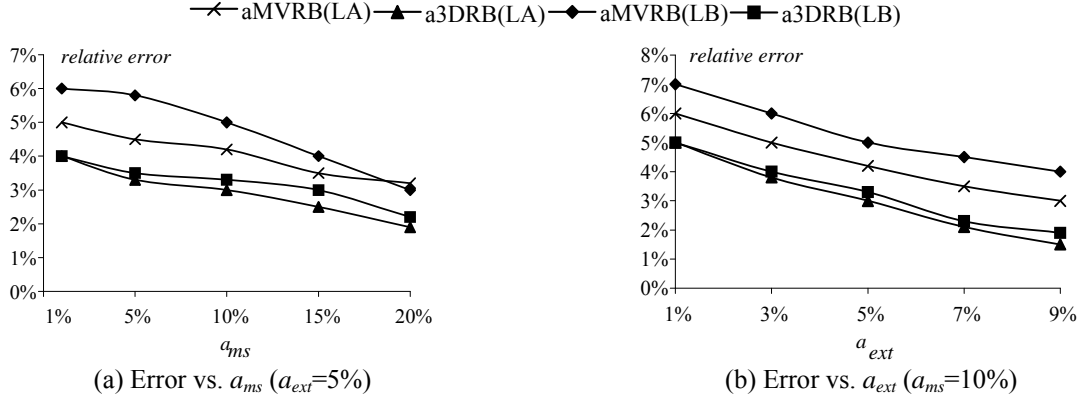


Fig.24. Size estimation accuracy (volatile regions)

Figure 25 demonstrates the error of Theorems 5.3, 5.5 (for query cost estimation) with respect to q_S , q_T , a_{ms} , and a_{ext} for *LA*. Comparing the diagrams in Figures 25 and 22, notice that the observation mentioned earlier also applies to volatile regions, i.e., the precision, in general, increases with the query overhead. Furthermore, the estimation of the query cost is less accurate than the size, as it accumulates the error of the histograms and the corresponding models for node extents. The results for *LB* are similar and omitted.

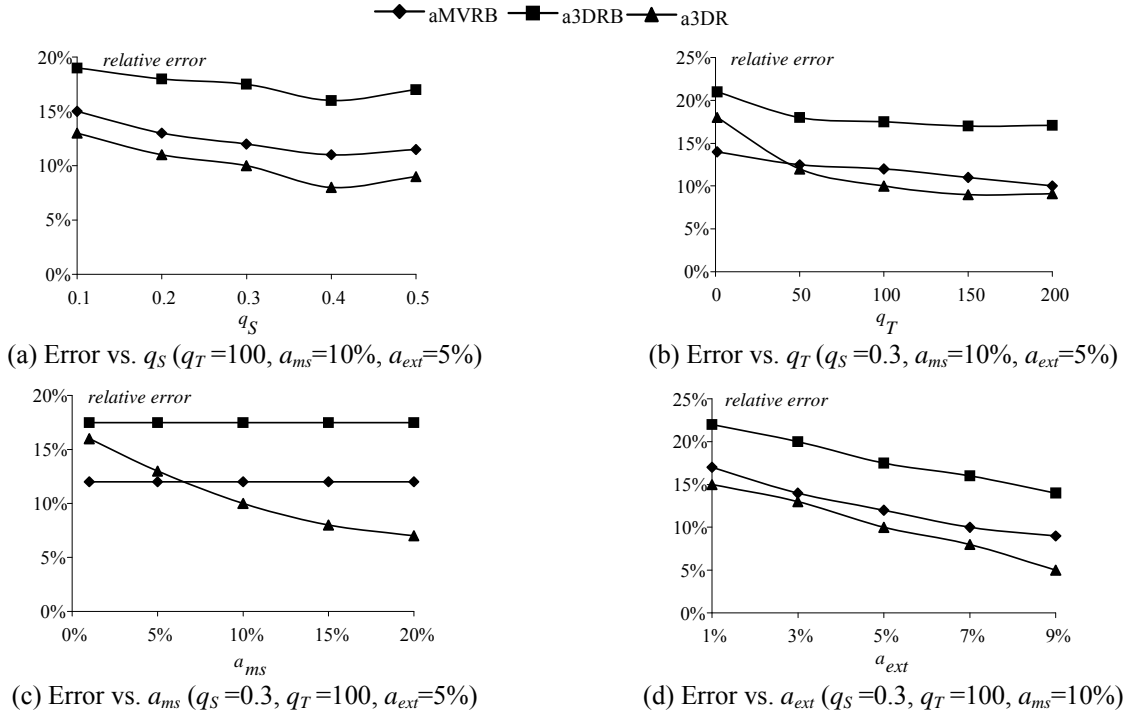


Fig.25. Node access estimation accuracy for *LA* (volatile regions)

To summarize, in this section we have experimentally confirmed the efficiency of the proposed structures for spatio-temporal aggregation. Specifically, for static regions, the *aRB-tree*, although consuming a fraction of the space required by the *a3DR-tree*, outperforms the *a3DR-tree* significantly in all cases. For volatile regions, the *a3DRB-tree* has the best overall performance in terms of size and query cost. Since, however, it is an off-line structure, the *aMVRB-tree* becomes the best alternative for applications requiring on-line indexing. In all cases, the traditional data

cube approach yields disappointing results. Finally, we also demonstrated that the proposed models can predict the performance accurately, incurring maximum error around 20% for real data distributions.

7. DISCUSSION AND CONCLUSION

Numerous real-life applications require fast access to summarized spatio-temporal information. Although data warehouses have been successfully employed in similar problems for relational data, traditional techniques have three basic impediments when applied directly in spatio-temporal applications: (i) no support for ad-hoc hierarchies, unknown at the design time, (ii) lack of spatio-temporal indexing methods, and (iii) limited provision for dimension versioning and volatile regions.

Here, we provide a unified solution to these problems by developing spatio-temporal structures that integrate indexing with the pre-aggregation technique. The intuition is that, by keeping summarized information inside the index, aggregation queries with arbitrary groupings can be answered by the intermediate nodes, thus saving accesses to detailed data. The applicability of our methods is demonstrated through a set of experiments that attempt to simulate realistic situations. In order to enable query optimization in practice, we also perform a comprehensive performance study for the existing and proposed structures, and present efficient cost models to capture the index sizes and query costs. Our results provide significant insights into the behavior of alternative methods, and analytically clarify the advantages of the proposed technique.

The proposed techniques can replace the data-cube in a star-schema-like implementation of spatio-temporal data-warehouses. Consider, for instance, the *aRB-tree* of Figure 5. Each leaf entry of the host R-tree can keep a pointer (foreign key) to the record storing information about the corresponding cell (e.g., phone company that owns the cell) in a table of regions. Given this dimension table, the system can answer queries of the form "find the total number of phone-calls (in cells intersecting q_R , during q_T) initiated by customers of Hong Kong Telecom". Similar pointers may be kept for the leaf entries of the B-trees, pointing to a dimension table with information about type of the timestamp (e.g., peak hour, cost of phone-calls) etc.

Although for simplicity we focused on the *sum* function, our techniques are directly applicable to multiple measures and functions. Consider, for instance, that queries inquire about the *maximum* number of phone-calls (during q_T) in some cell (intersecting q_R). Each intermediate entry r in the host and measure indexes must now store the maximum measure in its sub-tree (instead of the sum of measures). The query algorithms are exactly the same as in the case of *sum* for all the structures i.e., if the extent and lifespan of r is contained in q_R and q_T , its *max* value is aggregated directly and so on. In addition to distributive functions, the proposed techniques can also process *algebraic functions* (e.g., *average*), since they can be expressed as scalar functions of distributive functions (e.g., *sum/count*). Obviously, depending on the application needs, it is possible to have several measures (e.g., *sum* and *max*) associated with each entry. Furthermore, it is easy to devise processing algorithms for alternative query types such as: "for every cell in the city center (i.e., q_R) find the total number of phone calls in the last hour (i.e., q_T)". In this case the result contains several tuples, one for each cell qualifying the spatial condition (i.e., similar to a group-by). Query processing must now continue until the leaf level of the host index (the measures of intermediate entries are not aggregated), i.e., the

host index acts as a conventional spatio-temporal index.

A final note concerns the interpretation of the results of spatio-temporal aggregate queries, which depends on the application semantics of $R_i(t).ms$. If, for instance, $R_i(t).ms$ stores the number of mobile users (instead of initiated phone-calls) in region $R_i(t)$, the result should not be considered as the total number of users in q_R during q_T , since a user may be counted multiple times (if he/she stays in q_R for multiple timestamps). Tao et al [Tao et al. 2004] propose a method for duplicate elimination that combines spatio-temporal aggregation structures (e.g., the *aRB-tree*) with sketches [Flajolet and Martin 1985] based on probabilistic counting. Furthermore, note that our techniques, following the relevant literature [Jurgens and Lenz 1998, Papadias et al. 2001, Zhang et al. 2002, Govindarajan et al. 2003, Zhang et al. 2003], assume that if the query partially intersects a region, the entire measure of the region contributes to the query result. This is due to the fact that regions represent the highest resolution in the system. If additional information about the distribution of the objects within each region is available, we could take into account only the number of objects in the part of the region that intersects the query.

Spatio-temporal aggregation is a promising research area, combining various concepts of on-line analytical processing and multi-dimensional indexing, which is expected to play an important role in several emerging applications such as mobile computing and data streaming. A direction for future work includes supporting more complex spatio-temporal measures like the direction of movement. This will enable analysts to ask sophisticated queries in order to identify interesting numerical and spatial/temporal trends. The processing of such queries against the raw data is currently impractical considering the huge amounts of information involved in most spatio-temporal applications. Another topic worth studying concerns bulk updates, i.e., when a large number of regions issue updates synchronously (e.g., every timestamp). In this case instead of processing each update individually, we could exploit specialized bulk loading techniques adapted to the current problem.

ACKNOWLEDGEMENTS

A short version of this work appears in [Papadias et al. 2002]. We would like to thank Panos Kalnis and Jun Zhang for several discussions that led to this paper.

REFERENCES

- ACHARYA, S., POOSALA, V., AND RAMASWAMY, S. 1999. Selectivity Estimation in Spatial Databases. In *Proceedings of the ACM SIGMOD conference* (June), pp. 13-24.
- AGARWAL, P., ARGE, L., AND ERICKSON, J. 2000. Indexing Moving Points. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)* (May), pp. 175-168.
- BARALIS, E., PARABOSCHI, S., TENIENTE, E. 1997. Materialized View Selection in a Multidimensional Database. In *Proceedings of Very Large Database Conference (VLDB)* (August), pp. 156-165.
- BECKER, B., GSCHWIND, S., OHLER, T., SEEGER, B., WIDMAYER, P. 1996. An Asymptotically Optimal Multiversion B-Tree. *The VLDB Journal*, 5, 4, 264-275.
- 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 the ACM SIGMOD conference* (May), pp. 322-331.
- CHOI, Y., AND CHUNG, C. 2002. Selectivity Estimation for Spatio-Temporal Queries to Moving Objects. In *Proceedings of the ACM SIGMOD conference* (June), pp. 440-451.
- DENNY, M., FRANKLIN, M., CASTRO, P., PURAKAYASTHA, A. 2003. Mobiscope: A Scalable Spatial Discovery Service for Mobile Network Resources. In *Proceedings of the 4th Mobile Data Management (MDM)* (Jan.),

- pp. 307-324.
- FLAJOLET, P., MARTIN, G. 1985. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences*, 32, 2, 182-209.
- FORLIZZI, L., GÜTING, R., NARDELLI, E., SCHNEIDER, M. 2000. A Data Model and Data Structures for Moving Objects Databases. In *Proceedings of the ACM SIGMOD conference* (May), pp. 319-330.
- GAEDE, V., GÜNTHER, O. 1998. Multidimensional Access Methods. *ACM Computing Surveys*, 30, 2, pp. 123-169.
- GENDRANO, J., HUANG, B., RODRIGUE, J., MOON, B., SNODGRASS, R. 1999. Parallel Algorithms for Computing Temporal Aggregates. In *Proceedings of International Conference on Database Engineering (ICDE)*, pp. 418-427.
- GOVINDARAJAN, S., AGARWAL, P., ARGE, L. 2003. CRB-Tree: An Efficient Indexing Scheme for Range Aggregate Queries. In *Proceedings of International Conference on Database Theory (ICDT)* (Jan.), 143-157.
- GRAY, J., BOSWORTH, A., LAYMAN, A., PIRAHESH, H. 1996. Data Cube: a Relational Aggregation Operator Generalizing Group-by, Cross-tabs and Subtotals. In *Proceedings of International Conference on Database Engineering (ICDE)*, pp. 152-159.
- GUPTA, H. 1997. Selection of Views to Materialize in a Data Warehouse. In *Proceedings of International Conference on Database Theory (ICDT)* (Jan.), pp. 98-112.
- GUPTA, H., MUMICK, I. 1999. Selection of Views to Materialize Under a Maintenance-Time Constraint. In *Proceedings of International Conference on Database Theory (ICDT)* (Jan.), pp. 453-470.
- GÜTING, R., BÖHLEN, M., ERWIG, M., JENSEN, C., LORENTZOS, N., SCHNEIDER, M., VAZIRGIANNIS, M. 2000. A Foundation for Representing and Querying Moving Objects. *ACM Tran. Datab. Syst.*, 25, 1, 1-42.
- GUTTMAN, A. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD conference* (June), pp. 47-57.
- HADJIELEFThERIOU, M., KOLLIOS, AND G., TSOTRAS, V. 2003. Performance Evaluation of Spatio-Temporal Selectivity Estimation Techniques. In *Proceedings of Statistical and Scientific Database Management (SSDBM)* (July). pp. 202-211.
- HADJIELEFThERIOU, M., KOLLIOS, G., TSOTRAS, V., AND GUNOPULOS, D. 2002. Efficient Indexing of Spatiotemporal Objects, In *Proceedings of Extending Data Base Technology (EDBT)* (March). pp. 251-268.
- HAN, J., STEFANOVIC, N., KOPERSKI, K. 1998. Selective Materialization: An Efficient Method for Spatial Data Cube Construction. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)* (April), pp. 144-158.
- HARINARAYAN, V., RAJARAMAN A., ULLMAN, J. 1996. Implementing Data Cubes Efficiently. In *Proceedings of the ACM SIGMOD conference* (June), pp. 205-216.
- HURTADO, C., MENDELZON, A., VAISMAN, A. 1999. Maintaining Data Cubes under Dimension Updates. In *Proceedings of International Conference on Database Engineering (ICDE)* (March), pp. 346-355.
- JURGENS M., LENZ H. 1998. The Ra*-tree: An improved R-tree with Materialized Data for Supporting Range Queries on OLAP-Data. In *Proceedings of International Workshop on Database and Expert Systems Applications* (Aug.), pp. 186-191.
- KIMBALL, R. 1996. *The Data Warehouse Toolkit*. John Wiley.
- KLINE, N., SNODGRASS, R. 1995. Computing Temporal Aggregates. In *Proceedings of International Conference on Database Engineering (ICDE)* (March), pp. 222-231.
- KOLLIOS, G., GUNOPULOS, D., AND TSOTRAS, V. 1999. On Indexing Mobile Objects. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)* (May), pp. 261-272.
- KOLLIOS, G., GUNOPULOS, D., TSOTRAS, V., DELIS, A., HADJIELEFThERIOU, M. 2001. Indexing Animated Objects Using Spatiotemporal Access Methods. *Tran. Knowl. Data Eng. (TKDE)*, 13, 5, 758-777.
- KUMAR, A., TSOTRAS, V., FALOUTSOS, C. 1998. Designing Access Methods for Bitemporal Databases. *Tran. Knowl. Data Eng. (TKDE)*, 10, 1, 1-20.
- KWON, D., LEE, S., LEE, S. 2002. Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree. In *Proceedings of the 4th Mobile Data Management (MDM)* (Jan.), pp. 113-120.
- LAZARIDIS, I., MEHROTRA, S. 2001. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *Proceedings of the ACM SIGMOD conference* (June), pp. 401-412.
- LEE, M., HSU, W., JENSEN, C., CUI, B., TEO, K. 2003. Supporting Frequent Updates in R-Trees: A Bottom-Up Approach. In *Proceedings of Very Large Database Conference (VLDB)* (Sep.), pp. 608-619.
- MENDELZON, A., VAISMAN, A. 2000. Temporal Queries in OLAP. In *Proceedings of Very Large Database Conference (VLDB)* (Sep.), pp. 242-253.
- MOON, B., LOPEZ, I., IMMANUEL, V. 2000. Scalable Algorithms for Large Temporal Aggregation. In *Proceedings of International Conference on Database Engineering (ICDE)* (Feb.), pp. 145-154.

- PAGEL, B.U., SIX, H.W., TOBEN, H., WIDMAYER, P. 1993. Towards an Analysis of Range Query Performance in Spatial Data Structures. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)* (May), pp. 49-58.
- PAPADIAS, D., KALNIS, P., ZHANG, J., TAO, Y. 2001. Efficient OLAP Operations in Spatial Data Warehouses. In *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD)* (July), pp. 443-459.
- PAPADIAS, D., TAO, Y., KALNIS, P., ZHANG, J. 2002. Indexing Spatio-Temporal Data Warehouses. 2002. In *Proceedings of International Conference on Database Engineering (ICDE)* (Feb.), pp. 166-175.
- PFOSE, D., JENSEN, C, AND THEODORIDIS, Y. 2000. Novel Approaches to the Indexing of Moving Object Trajectories. In *Proceedings of Very Large Database Conference (VLDB)* (Sep.), pp. 395-406.
- SALTENIS, S., JENSEN, C. 2002. Indexing of Moving Objects for Location-Based Services. In *Proceedings of International Conference on Database Engineering (ICDE)* (Feb.), pp. 463-472.
- SALTENIS, S., JENSEN, C., LEUTENEGGER, S., AND LOPEZ, M. 2000. Indexing the Positions of Continuously Moving Objects. In *Proceedings of the ACM SIGMOD conference* (June), , pp. 331-342.
- SALZBERG, B., TSOTRAS, V. 1999. A Comparison of Access Methods for Temporal Data. *ACM Computing Surveys*, 31, 2, 158-221.
- SHUKLA, A., DESHPANDE, P., NAUGHTON, J. 1998. Materialized View Selection for Multidimensional Datasets. In *Proceedings of Very Large Database Conference (VLDB)* (Aug.), pp. 488-499.
- SISTLA, A., WOLFSON, O., CHAMBERLAIN, S., DAO, S. 1997. Modeling and Querying Moving Objects. In *Proceedings of International Conference on Database Engineering (ICDE)* (April), pp. 422-432.
- STEFANOVIC, N., HAN, J., KOPERSKI, K. 2000. Object-Based Selective Materialization for Efficient Implementation of Spatial Data Cubes. *Tran. Knowl. Data Eng. (TKDE)*, 12, 6, 938-958.
- TAO, Y., KOLLIOS, G., CONSIDINE, J., LI, F., PAPADIAS, D. 2004. Spatio-Temporal Aggregation Using Sketches. In *Proceedings of International Conference on Database Engineering (ICDE)* (March), pp. 214-226.
- TAO, Y., PAPADIAS, D. 2001. The MV3R-tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proceedings of Very Large Database Conference (VLDB)* (Sep.), pp. 431-440.
- TAO, Y., PAPADIAS, D. 2004. Performance Analysis of R*-Trees with Arbitrary Node Extents. *Tran. Knowl. Data Eng. (TKDE)*, 16, 6, 653-668.
- TAO, Y., PAPADIAS, D., AND SUN, J. 2003a. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. *Proceedings of Very Large Database Conference (VLDB)* (Sep.), pp. 790-801.
- TAO, Y., PAPADIAS, D., ZHANG, J. 2002a. Cost Models for Overlapping and Multi-Version Structures. *ACM Tran. Datab. Syst.*, 27, 3, 299-342.
- TAO, Y., PAPADIAS, D., ZHANG, J. 2002b. Efficient Processing of Planar Points. In *Proceedings of Extended Database Technology (EDBT)* (March), pp. 682-700.
- TAO, Y., SUN, J., PAPADIAS, D. 2003b. Selectivity Estimation for Predictive Spatio-Temporal Queries. *ACM Tran. Datab. Syst.*, 28, 4, 295-336.
- THEODORIDIS, Y., SELLIS, T. 1996. A Model for the Prediction of R-tree Performance. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)* (June), pp. 161-171.
- THEODORIDIS, Y., STEFANAKIS, E., SELLIS, T. 2000. Efficient Cost Models for Spatial Queries Using R-Trees. *Tran. Knowl. Data Eng. (TKDE)*, 12, 1, 19-32.
- TIGER. <http://www.census.gov/geo/www/tiger/>
- VARMAN, P., VERMA, R. 1997. Optimal Storage and Access to Multiversion Data. *Tran. Knowl. Data Eng. (TKDE)*, 9, 3, 391-409.
- VAZIRGIANNIS, M., THEODORIDIS, Y., SELLIS, T. 1998. Spatio-Temporal Composition and Indexing for Large Multimedia Applications. *Multimedia Systems*, 6, 4, 284-298.
- YANG, J., WIDOM, J. 2003. Incremental Computation and Maintenance of Temporal Aggregates. *The VLDB Journal*, 12, 3, 262-283.
- YAO, S. Random 2-3 Trees. 1978. *Acta Informatica*, 2, 9, 159-179.
- ZHANG, D., GUNOPULOS, D., TSOTRAS, V., SEEGER, B. 2002. Temporal Aggregation Over Data Streams Using Multiple Granularities. In *Proceedings of Extended Database Technology (EDBT)* (March), pp. 646-663.
- ZHANG, D., GUNOPULOS, D., TSOTRAS, V., SEEGER, B. 2003. Spatial and Temporal Aggregation Over Data Streams Using Multiple Granularities. *Information Systems*, 28, 1-2, 61-84.
- ZHANG, D., MARKOWETZ, A., TSOTRAS, V., GUNOPULOS, D., SEEGER, B. 2001. Efficient Computation of Temporal Aggregates with Range Predicates. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)* (May), pp. 237-245.
- ZHANG, D., TSOTRAS, V., GUNOPULOS, D. 2002. Efficient Aggregation over Objects with Extent. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)* (May), pp. 121-132.

Received May 2003; revised May 2004; accepted August 2004.