Spatiotemporal Aggregate Computation: A Survey

Inés Fernando Vega López, Richard T. Snodgrass, *Senior Member*, *IEEE*, and Bongki Moon

Abstract—Spatiotemporal databases are becoming increasingly more common. Typically, applications modeling spatiotemporal objects need to process vast amounts of data. In such cases, generating aggregate information from the data set is more useful than individually analyzing every entry. In this paper, we study the most relevant techniques for the evaluation of aggregate queries on spatial, temporal, and spatiotemporal data. We also present a model that reduces the evaluation of aggregate queries to the problem of selecting qualifying tuples and the grouping of these tuples into collections on which an aggregate function is to be applied. This model give us a framework that allows us to analyze and compare the different existing techniques for the evaluation of aggregate queries. At the same time, it allows us to identify opportunities for research on types of aggregate queries that have not been studied.

Index Terms—Spatiotemporal databases, aggregation queries, aggregate function.

1 Introduction

A wide variety of scientific and business applications need to capture the spatial and time-varying characteristics of the entities they model. Spatial, temporal, and spatiotemporal applications are becoming more common with the increasing capabilities of computer systems to store and process large amounts of information. Examples of such applications include land management, weather monitoring, natural resources management, environmental, ecological, and biodiversity studies, tracking of mobile devices, and navigation systems.

Typically, spatiotemporal applications store vast amounts of data. For example, remotely sensed data from NASA is captured at a rate of several Gigabytes a day. Clearly, due to the size of the data sets, the study of individual entries in the database is rarely feasible, and, in some cases, not possible for legal reasons (i.e., keeping track of the trajectory followed by a cell phone user). In addition, as data sets grow larger, there is a need for extracting general characterizations of large subsets of the data. Therefore, it is useful to develop techniques that efficiently summarize and discover trends in data and help in decision making. For example, in a traffic control application, rather than studying the precise position of every single vehicle in a particular road, it may be of interest just to know the overall number of cars crossing an intersection during rush hour. This summarized information may support decisions regarding the construction of new roads and underpasses or the addition of new traffic lights, for instance. Similarly, for biodiversity studies, it might be of interest to determine the distribution of taxa in a particular region, rather than the specific position of each plant or animal. Once a region rich in biodiversity has been detected, it can be considered for declaration of a natural reserve.

In recent years, there has been an increasing amount of research work dedicated to spatiotemporal data. Efforts have focused on identifying relevant characteristics of spatiotemporal entities and in providing models for this new kind of data [2], [12], [22], [58], [67], [78]. We have also observed interest in the organization of data for efficient retrieval [32], [57], [65]. The goal of this paper is to provide a survey on state-of-the-art techniques for computing spatiotemporal aggregate functions, a topic of fervent interest during the last few years. (An example spatiotemporal aggregate is: For every county in the state of Arizona, what has been the yearly forest density for the last 10 years? Note that both the underlying data and the result vary over space and time.)

Aggregate functions are widely used in database applications. Their popularity is reflected in the presence of aggregates in a large number of queries in the decision support benchmark *TPC-D* [28]. The ability of aggregate functions to provide summarized information from a large collection of data is indeed fundamental in specific, increasingly relevant, application domains such as On Line Analytical Processing (OLAP), decision support, statistical evaluation, and management of geographical data [9], [11], [14], [29], [33], [83].

In this paper, we decompose a spatiotemporal object into the different extents associated with it, namely, its explicit attributes, its spatial extent, its temporal extent, and the combination of spatial and temporal extents. We similarly characterize all the distinct types of spatiotemporal aggregation queries that can be of interest for a given user. This characterization allows us to classify different approaches in the literature within a common framework. The rest of this survey is organized as follows: Section 2 gives the preliminaries on spatiotemporal objects and aggregation. The following three sections present the semantics of aggregate functions for traditional relational databases, temporal databases, and spatial databases, along with most techniques for computing aggregates on such databases. Section 6 is

- *I.F. Vega López is with the Escuela de Informatica, Universidad Autonoma de Sinaloa, Josefa Ortiz S/N, Ciudad Universitaria, Culiacan, Sinaloa, 8000, Mexico. E-mail: ifvega@uas.uasnet.mx.*
- *R.T. Snodgrass and B. Moon are with the Department of Computer Science, University of Arizona, PO Box 210077, Tucson, AZ 85721-0077. E-mail: {rts, bkmoon}@cs.arizona.edu.*

Manuscript received 4 Dec. 2003; revised 16 June 2004; accepted 6 July 2004; published online 17 Dec. 2004.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0250-1203.

TABLE 1
Employees: A Sample Relation

$Name$	$DepartmentId$	Age	$Salary$
Praveen	1	24	45000
John	1	28	42000
Frank	1	50	80000
Sophia	2	25	40000
Fernando	2	30	48000

devoted to spatiotemporal aggregation. Opportunities for research are presented in Section 7. Finally, Section 8 summarizes the state of the techniques described in this paper.

2 Preliminaries

In studying previous research on aggregation, we have observed inconsistencies in the problem definition, as well as in the terms used to refer to specific concepts. Therefore, we start with a concrete definition of the problem of aggregation on databases and present a model to describe aggregation queries. Such a model can be applied to traditional databases in which objects lack temporal or spatial extent, as well as to temporal, spatial, and spatiotemporal databases. This allows us to classify previous work on aggregation and to identify areas of this problem that have not yet been addressed by the research community.

By definition, a *spatiotemporal* object is a unified object with spatial and temporal extent [12], [82]. A pure spatial object can be either a point, a line, or a region in two or three-dimensional space [21], [46], [80]. If no spatial information is required, no spatial extent is associated to the object. The time extent of an object can be modeled by either valid time, transaction time, or both (in which case, the object is known as a bitemporal object) [19]. Of course, the time variation of the objects being modeled may be of no relevance, in which case, the temporal extent of the object is simply ignored. When the time-varying behavior of a spatial object is of interest, we have objects that change position, shape, and both position and shape [78]. Therefore, the combination of the temporal extent with the spatial extents of an object leads to different models, ranging from snapshots to three-dimensional, four-dimensional, and even five-dimensional objects depending on the different definitions of time and space [35], [82].

2.1 Aggregate Functions

An *aggregate function* takes a set of tuples and returns a single value that summarizes the information contained in the set of tuples [9], [29], [42]. In the context of this survey, *aggregation* is the effect of applying an aggregate function to a group of qualifying tuples. Aggregate functions have received several names in the extent research literature. Epstein differentiates between *scalar aggregate* and *aggregate function* to distinguish between queries returning single or multiple aggregate values [20]. In Epstein's work, an aggregate query can return several results if the tuples are first partitioned into disjoint subsets based on a grouping attribute (i.e., a `GROUP-BY` query). The SQL standard uses the term *set functions* to refer to aggregate functions. In the rest of this paper, we refer to the functions computing an aggregate value from a set of tuples as aggregate functions.

TABLE 2
The Result of an Aggregate Query Using Group Composition and the `MAX` Aggregate Function

$DepartmentId$	MAX($Salary$)
1	80000
2	48000

As we will see, there is no need to differentiate between functions returning either single or multiple aggregate values. Multiple aggregate values are the result of an orthogonal operator that partitions the relation into sets of tuples also known as aggregation groups. We favor the term *aggregate function* over *set function* used by SQL because it is more specific.

The SQL standard provides a variety of aggregate functions. The SQL-92 standard includes five such functions, namely, `COUNT`, `SUM`, `AVG`, `MIN`, and `MAX` [50]. The SQL:1999 standard adds `EVERY`, `SOME`, and `ANY`, whereas the SQL/OLAP addendum[1] to the SQL:1999 standard includes 18 additional aggregate functions [50].

2.2 Aggregation on Explicit Attributes

Aggregate functions are applied to a collection (i.e., set) of tuples. Given a relation, a collection of tuples can be generated at three different levels. For example, consider the relation **employees** given in Table 1, whose schema is $\{Name, DepartmentId, Age, Salary\}$. A typical aggregate query on this relation could be the following.

Query 1. *Compute the highest salary on each department.*

The results for Query 1 are shown in Table 2. In this case, the `MAX` aggregation function is applied to collections of tuples created by a process known as *group composition*. In group composition, tuples sharing the same values in a list of attributes (termed *grouping attributes*) form a collection. Because these collections of tuples result from grouping composition, let us call them *groups*. Aggregate functions are then applied to each group of tuples. This procedure aggregates information at a coarse level because a single aggregate value is generated for each group. In the case of Query 1, two groups of tuples were generated by the distinct values of $DepartmentId$ (i.e., the grouping attribute).

Different queries may request to generate aggregate values at a finer level. Consider, for example, the following query.

Query 2. *Within each department, compute the highest salary by age. For each different value of age, consider the salary of the next youngest employee.*

The results for Query 2 are shown in Table 3 (for brevity, only the results for the first department are shown). In this case, the `MAX` aggregate function is applied to collections of tuples generated by a process known as *partition composition*. During *partition composition*, tuples sharing the same values in a list of attributes (termed *partition attributes*) are placed in the same

1. The aggregate functions defined in SQL/OLAP are `STDDEV_POP`, `STDDEV_SAMP`, `VAR_POP`, `VAR_SAMP`, `COVAR_POP`, `COVAR_SAMP`, `CORR`, `REGR_SLOPE`, `REGR_INTERCEPT`, `REGR_COUNT`, `REGR_R2`, `REGR_AVGX`, `REGR_AVGY`, `REGR_SSX`, `REGR_SSY`, `REGR_SYY`, `PERCENTILE_DISC`, and `PERCENTILE_CONT`.

TABLE 3
The Result of an Aggregate Query Using Partition Composition on $DepartmentId$, Sliding Window Composition on Age, and MAX Aggregate Function on $Salary$

$Name$	$DepartmentId$	Age	$Salary$	$MAX(Salary)$
Praveen	1	24	45000	45000
John	1	28	42000	45000
Frank	1	50	80000	80000

collection. Because these collections result from *partitioning composition*, let us call them *partitions*. For Query 2, partition composition has been defined on $DepartmentId$, resulting in two partitions. Aggregate functions are not applied directly to partitions. Instead, *sliding window composition* is performed on each partition. During sliding window composition, a *window frame* is placed around each tuple in the partition. A window frame is defined using a range of values (logical size) or a number of tuples (physical size), either leading or trailing (or both) each tuple in the partition. In the case of Query 2, the window frame was defined as "1 tuple trailing." This effectively selects a set of tuples on which an aggregate function is applied. For example, for the first tuple in the partition (i.e., Praveen), there are no tuples trailing. Hence, the aggregate function is applied only to the current value of $Salary$ (45,000 in this case). For the second tuple in the partition (i.e., John), the window frame selects the current and previous tuples in the partition. The MAX aggregate function is then applied to the set $\{45,000, 42,000\}$, resulting in the aggregate value 45,000. For the third tuple in the partition, the aggregate function is applied to the set $\{42,000, 80,000\}$, yielding the aggregate value 80,000. Note that, in this case, we generate an aggregate value for every tuple in every partition of every group. If group composition is not used (such as in Query 2), the entire relation is considered as a single group.

To complement the aggregate functions, the SQL standard includes mechanisms for defining collection of tuples at these three levels. The SQL-92 standard includes the `GROUP BY` clause to perform grouping composition. The need for partitioning and sliding window composition was later noted and the SQL:1999 standard includes the `WINDOW` clause to address both of these needs.

2.3 Aggregation on the Temporal and Spatial Extent

Similar to the case for explicit (nontemporal and nonspatial) attributes, the implicit attributes of a spatiotemporal object can be used to define collections of tuples on which to apply aggregate functions. For the temporal dimension, these collections are defined by a process called *temporal grouping* [40], in which the time line is partitioned and tuples are grouped over these time partitions. Temporal aggregation, as studied in the literature, uses time granularities as the building blocks for temporal grouping. Different granularities of the time dimension can be used to define *temporal group composition*, *temporal partition composition*, and *temporal sliding widow composition*. Details on how this is achieved, along with illustrative examples, will be presented in Section 4.

Similar to temporal grouping, *space grouping* is the process of defining collections of tuples based on a partition of space. Different levels of spatial granularities can be used to define *spatial group composition*, *spatial partition composition*, and *spatial sliding window composition* in a spatial relation. Further details and examples will be presented in Section 5.

The temporal and spatial extents of spatiotemporal objects are orthogonally defined. Therefore, the concept of aggregation groups can be defined independently on each dimension and should not affect our formalism. The reader is referred to Section 6 for a detailed description of spatiotemporal aggregation.

3 Aggregate Functions on Explicit Attributes

Computing aggregations has always been considered an important feature of practical database query languages. An *aggregate query* is a query involving *aggregate functions* and it usually includes predicates and other operators to select and reorganize qualifying tuples from the database.

Aggregate functions produce a single value over a collection of qualifying tuples from a relation [9], [29], [42]. As we have mentioned in Section 2.2, these collections of tuples can be defined using group composition (e.g., the `GROUP-BY` clause in SQL) and partition and sliding window composition (e.g., the `WINDOW` clause in SQL). Klug [42] provided a formal framework for defining aggregate functions for relational databases. In his model, for a relation with n attributes, he proposed using a set of n aggregate functions, each function defined on one attribute of the relation. Formally, for a relation R with schema $\{A_1, A_2, \ldots, A_n\}$, with each attribute A_i associated with domain D_i, a countable set $Agg = \{f_1, f_2, \ldots, f_n\}$ of aggregate functions should exist. Each function $f_i \in Agg$ operates on attribute $A_i \in R$ and, for each function $f_i \in Agg$, $f_i : D_1 \times D_2 \times \ldots D_n \rightarrow D_{agg}$, where D_{agg} is the domain of the aggregate function. Note that D_{agg} can be different from D_i. For instance, consider the AVG aggregate function. In such a case, while D_i can be the set of integers, D_{agg} will correspond to the set of reals. Here, we present a framework for analyzing aggregate queries based on the mechanisms used to define collections of tuples.

3.1 Formal Definition of Aggregation on Explicit Attributes

The aggregation functions defined by the SQL-92 standard can be evaluated as indicated in Table 4. Each of these functions operates over a virtual relation. This virtual relation is a collection of tuples defined by group composition, partition composition, and sliding window composition. Let us consider an aggregate query using group composition such as Query 1 presented in the previous section. In this query, tuples were first assigned to collections based on the value of their attribute $DepartmentId$. Then, an aggregate function (e.g., MAX) was applied to each collection. Formally, an aggregate query using group composition generates an aggregate value for each resulting group as follows.

Definition 1 (Aggregation Using Group Composition). *Given an aggregation query on relation R and a select predicate SP, using group composition to define collections of*

TABLE 4
Evaluation of the SQL-92 Aggregate Function

Aggregate	Evaluation	Evaluation with unique values
$COUNT_i(R)$	$\|R\|$	$\|\pi_{A_i}(R)\|$
$SUM_i(R)$	$\sum(\{r.A_i \| r \in R\})$	$\sum(\{r.A_i \| r \in \pi_{A_i}(R)\})$
$AVG_i(R)$	$SUM_i(R)/COUNT_i(R)$	$SUM_i(\pi_{A_i}(R))/COUNT_i(\pi_{A_i}(R))$
$MIN_i(R)$	$min(\{r.A_i \| r \in R\})$	$min(\{r.A_i \| r \in R\})$
$MAX_i(R)$	$max(\{r.A_i \| r \in R\})$	$max(\{r.A_i \| r \in R\})$

tuples based on the values of attribute list A of R, the solution to the query can be generated as follows:

Let S be the set of distinct values contained in the attribute list A. That is, $S = \pi_A(R)$. Every $s \in S$ partitions the value domain of A and generates groups of tuples from R as

$$G_{A,SP}(s,R) = \{r | r \in R \wedge r[A] = s[A] \wedge SP(r)\}. \qquad (1)$$

Now, the solution to an aggregate query using the groups of tuples defined by S over relation R, $S = \pi_A(R)$, is given by the expression

$$GAgg_{f_i,A,SP}(R) = \{s \circ f_i(G_{A,SP}(s,R)) | s \in \pi_A(R)\},$$

where f_i is the aggregate function. This query produces a new relation whose schema is $A \cup Agg$.

Let us now consider an aggregate query using partition composition such as Query 2 presented in the previous section. As we have seen, this query generates collections of tuples based on the values of a list of attributes, but, instead of generating a single aggregate value for each collection, an aggregate value for every tuple in the collection is generated. Therefore, there is a need to define a window to slide through all tuples in the collection. An aggregate function is then applied to the set of tuples covered by the window. Formally, an aggregate query using partition composition generates an aggregate value for each tuple in the relation as follows:

Definition 2 (Aggregation Using Partition Composition). *Given an aggregation query on relation R and a select predicate SP, using the partition composition to define partitions of tuples based on the values of attribute list A of R and sliding window composition based on a single attribute B of R, the solution to the query can be generated as follows:*

Let the list of attributes A of R create a data partition P as defined by (1). During sliding window composition, for each tuple p in P, a window frame is defined by the following expression:[2]

$$WF_{precedes,follows,B}(p,P) = \{t | t \in P \wedge (p[B] - precedes) \leq t[B] \leq (p[B] + follows)\}.$$

In the expression, p[B] gives the value of attribute B of p and precedes and follows are query arguments. Now, the solution to an aggregate query using data partitions defined by A over relation R, window fames defined on attribute B of R, and a range on B defined by precedes and follows, is given by the following expression:

$$WAgg_{f_i,A,SP,B,precedes,follows}(R) = \{p \circ f_i(WF_{precedes,follows,B}(p,P)) | p \in P_{A,SP}(s,R) \wedge s \in \pi_A(R)\}$$

The resulting relation has schema $R \cup Agg$.

We can evaluate Query 2 using this semantics. For this, we simply need to set the values of *precedes* and *follows* to 1 and 0, respectively. The sliding window is defined on attribute *Age* (i.e., $B = Age$). Similarly, the list of partition attributes $A = \{DepartmentId\}$ and $f_i = \texttt{MAX}_{Salary}$.

3.2 Existing Approaches for Evaluating Aggregate Queries

A simple two-step algorithm was proposed by Epstein for evaluating aggregate queries [20]. To handle many aggregate functions in a query, the algorithm computes each of them separately and stores each result in a singleton relation, referring to that singleton relation when evaluating the rest of the query. A different approach employing program transformation methods was proposed by Freytag and Goodman to systematically generate efficient iterative programs for aggregate queries [23]. For brevity, we omit further details on these methods because they are not critical to understanding spatiotemporal aggregation.

Recently, research work has explored diverse aspects of the aggregation operation. Among them are **optimization**, for applications where performance is of utter importance [44], [83], **online aggregation**, where the user is aware of the progress made by the query processor and he/she is capable of stopping the query once an acceptable result has been achieved [31], [33], or **approximate** solutions, for applications where an exact solution is not required and a fast *good* answer is preferred [10], [11], [26], [27]. These are techniques that can be applied to the computation of aggregate functions in general. We provide more detail whenever these techniques are presented as part of the existing approaches for evaluating temporal, spatial, or spatiotemporal aggregation.

3.3 Aggregation and OLAP

Typical OLAP queries aggregate data across several attributes (i.e., columns) in a relation. The CUBE operator [29], for instance, was proposed as the *n*-dimensional generalization of the `GROUP-BY` clause in SQL. It computes `GROUP-BY` s corresponding to all possible combinations of a list of attributes. This implies finding the power set of all attributes in the relation, which is not a trivial task. Thus, solving aggregate queries in OLAP applications has inspired a considerable amount of research work. A general assumption in the CUBE operator is that the aggregate function being computed is distributive. Therefore, aggregate functions can be partially computed on disjoint subsets of data. By precomputing the aggregated results of different

2. While the SQL standard contemplates the possibility of defining window frames by specifying its physical size, this implies having some ordering in the tuples. This is not possible using set algebra. Therefore, we do not contemplate this possibility in our model.

subsets of data, the total processing time of a query can be drastically reduced. The final result can be obtained by properly merging these partial results [3], [14], [34].

While the different columns in a data cube are usually called "dimensions," they generally cannot be considered as dimensions in a spatial database. This is because some of the dimensions in a data cube (e.g., *CustomerId*) are defined over discrete domains which do not have a natural ordering among their values (customer $1,000$ cannot be considered "close" to customer $1,001$). In such cases, any ordering defined for the values in one of these columns is arbitrary. For this reason, we differentiate databases for OLAP applications from spatial databases. For the same reason, we do not consider these "dimensions" as a special extent of the entities modeled by the database; instead, they can be regarded as explicit attributes that characterize a particular entity.

4 Temporal Aggregates

While a conventional database models the reality relevant to an enterprise as a single state, a temporal database is one that supports some aspect of time and keeps track of the different states of the database. Time-varying data is common and applications that manage such data abound [6], [49], [86]. In a temporal database, the temporal data is modeled as a collection of line segments. These line segments have a begin time, an end time, one or more time-invariant attributes, and one or more time-varying attributes. It is well-known that database facts have at least two relevant temporal aspects. *Valid time* concerns when a fact was true in the modeled reality. *Transaction time*, on the other hand, concerns when a fact was current in the database. These two aspects are orthogonal in that each could be independently recorded or not and each has associated with it specific properties [6], [36], [68], [69]. All methods to date have focused on one time dimension only. However, most of them can be easily extended to handle either valid or transaction time.

4.1 Formal Definition of Temporal Aggregation

Computing temporal aggregates is a significantly more intricate problem than conventional aggregation because each database tuple is accompanied by a time interval during which its attribute values are valid. Consequently, the value of a tuple attribute affects the aggregate computation for all those instants included in the tuple's time interval.

In traditional databases, where only explicit attributes are of concern, aggregate functions are applied to collections of tuples that are defined by the different values in a list of explicit attributes. For the temporal extent of an object, collections of tuples can be defined based on time granularities (such characterization will allow the approaches we discuss below to be classified).

A time domain is the set of primitive temporal entities used to define and interpret time-related concepts [18], [54]. Formally, a time domain is a totally ordered set of time points with the ordering relation $\leq$. A granularity creates a discrete image, in terms of *granules*, of the time domain. Portions of the time-domain are grouped into aggregations called *granules*. A granule is a subset of the time domain. A granularity is a mapping G from the integers to granules. Granularities are related in the sense that the granule in one granularity may be further aggregated to form larger granules belonging to a coarser granularity [7], [8], [18].

Temporal group composition is a mechanism that generates collections of tuples. A collection, termed a group, is formed by all tuples valid at the same time value at granularity G. An aggregate function can then be applied to each group. *Temporal partition composition* is used for handling queries that require aggregation at a finer level. Temporal partition composition defines collections of tuples, termed partitions, based on the distinct time values at granularity H (H *is finer than* G, denoted by $H \prec G$). However, aggregation functions are not applied to these partitions. Instead, *temporal sliding window composition* places window frames around each time value at granularity J ($J \prec H$) within these partitions. A window frame is defined by a time interval leading, trailing, or leading and trailing every time value in the partition. The aggregate function is applied to the set of tuples valid for the window frame around each time value within a partition.

The generation of collections of tuples based on some partition of the time domain has received several names in the research literature. In particular, we have encountered the terms *span grouping* and *instant grouping*. For span grouping, the timeline is partitioned in predefined intervals such as year, month, or day [40]. Instant grouping, on the other hand, is defined by the data [40], [53], [70]. These two are really special cases of temporal group composition. In the former, the granularity used is that of a year, month, day, etc. In the latter, the granularity used for temporal group composition is the finest granularity supported by the temporal relation.

When computing temporal aggregation using group composition, the resulting relation is a time-varying relation defined at granularity G (i.e., the granularity of the groups). Consider, for example, the following query.

Query 3. *Compute the monthly average salary of all employees.*

In this query, the timeline is partitioned using fixed intervals (i.e., months). Groups of tuples are defined by each temporal partition and the aggregation function is applied to each group. This kind of aggregation query is formally defined as follows:

Definition 3 (Aggregation Using Temporal Group Composition). *Given a temporal relation R^T and a select predicate SP, let $\mathcal{T}(G, R^T) = \{\tau | \tau \in cast(r[vt], G) \wedge r \in R^T\}$, where $r[vt]$ gives the valid time of r, be the set of time values at granularity G for which there is at least one tuple in the temporal relation R^T that is valid at that time value and at that granularity. The function $cast(r[vt], G)$ returns the time values at granularity G that contain $r[vt]$. Each time value $\tau \in \mathcal{T}(G, R^T)$ defines a collection of tuples in R^T based on a time partition as follows:*

$$\begin{aligned} P_{G,SP}(\tau, R^T) = \{t \mid \exists\, r \in R^T \wedge overlaps(cast(r[vt], G), \tau) \\ \wedge SP(r) \wedge \ (t[A_1, \dots, A_n] = r[A_1, \dots, A_n]) \\ \wedge (t[vt] = intersect(r[vt], \tau)\}. \end{aligned} \tag{2}$$

The result of an aggregate query using temporal group composition with granularity G is given by the following expression:

$$GBAgg_{f_i,G,SP}(R^T) = \{\tau \circ f_i(P_{G,SP}(\tau, R^T)) | \tau \in \mathcal{T}(G, R^T)\}.$$

In (2), $r[A_1, \ldots, A_n]$ refers to the explicit attributes of tuple $r \in R^T$. Note that P, as defined by (2), does not generate a strict partition of R^T such as the case of (1). Instead, P is a subset of the rows in R^T. The temporal extent of the tuples in P has been narrowed to a single granule in granularity G. Also note that, in order to provide a clean and simple notation, (2) generates groups based on the implicit attribute *valid time*. This definition can be easily modified to account for *transaction time*.

For an aggregation using temporal partition composition, on the other hand, there are at least two granularities involved. The first (i.e., coarser) granularity defines collections of tuples called partitions, whereas the second (i.e., finer) granularity is used to define window frames during temporal sliding window composition. When temporal partition composition at granularity H is used in combination with temporal group composition at granularity G, $H \prec G$. For example, consider the following query requiring temporal partition composition and temporal window composition.

Query 4. *For every year, compute the moving average of salary of all employees with respect to the previous two months.*

In this case, a temporal partition is defined at the granularity level of year. Within each partition, tuples are grouped by month. All tuples valid at a particular month and the previous two months form a group on which the aggregate function is to be applied. The result of the query is an aggregate value for every time value at the granularity level of month. We can formally define this kind of query as follows:

Definition 4 (Aggregation Using Temporal Partition Composition). *Given a temporal relation R^T and a select predicate SP, let us use $\mathcal{T}(H, R^T)$ and $P_{H,SP}(\tau, R^T)$ as before. Let J be a granularity such that $J \prec H$. For each $t \in \mathcal{T}(J, R^T)$, a window frame with respect to the time partition defined by granularity H is generated as*

$$WF_{H,SP,J,precedes,follows}(t, R^T) = \{r | r \in P_{H,SP}(cast(t, H), R^T) \wedge overlaps(cast(r[vt], J), [t - precedes, t + follows))\},$$

where precedes and follows are query arguments that define the aggregation group around each time value t within a window partition.

The result of an aggregate query on temporal relation R^T with temporal partitions at granularity H and window frames at granularity J with ranges defined by precedes and follows is given here.

$$WAgg_{f_i,H,SP,J,precedes,follows}(R^T) = \{cast(t, H) \circ t \circ f_i(WF_{H,SP,J,precedes,follows}(t, R^T)) | t \in \mathcal{T}(J, R^T)\}.$$

For Query 4, we have used the granularity level $year$ to define partitions ($H = year$) and the granularity level $month$ to define window frames ($J = month$). The function $cast(t, H)$ obtains the time value for the granularity level $year$ from a particular month. For instance, $cast(06/1973, year)$ will generate the time value 1973. A window frame is defined by including the previous two temporal values at granularity level $month$ (i.e., $trailing = 2$), but no future values (i.e., $leading = 0$). For all tuples valid within a particular window frame, the AVG aggregate function is applied.

4.2 Existing Approaches for Evaluating Temporal Aggregate Queries

Various algorithms have been proposed for processing temporal grouping and computing aggregation on a temporal relation. These algorithms can be classified based on the time when the aggregate value is computed. *Nonindexed evaluation* algorithms scan the temporal relation every time an aggregate query is issued. During this process, collections of tuples are generated based on temporal grouping and an aggregate function is applied to each collection. *Indexed evaluation* algorithms, on the other hand, preprocess aggregate values and store this information in a disk-based data structure. Instead of scanning the temporal relation when a query is issued, indexed evaluation algorithms use this data structure for answering an aggregation query.

4.2.1 Nonindexed Aggregation Evaluation

The earliest approach for evaluating temporal aggregation was proposed by Tuma [79]. He proposed a two-step algorithm. In the first step, the temporal relation is scanned once to determine *constant intervals*. A constant interval is a period for which the temporal relation remains unchanged [70]. In the second step, the temporal relation is scanned again to apply the aggregate function on the groups of tuples defined by the constant intervals. Tuma's approach is based on Epstein's [20] algorithm for computing aggregation over explicit attributes using the GROUP-BY operator.

I/O efficient algorithms for computing temporal aggregation were developed after Tuma's initial approach. These methods require reading the temporal relation only once. A data structure (usually maintained in main memory) is created as tuples in the temporal relation are processed. The resulting data structure holds sufficient information to compute temporal aggregation.

The Aggregation Tree: Kline and Snodgrass [40] proposed an algorithm for computing temporal aggregation using a main memory-based data structure. The proposed algorithm was called *aggregation tree* because it builds a tree while scanning a temporal relation. After the tree has been built, the answer to the temporal aggregation query is obtained by traversing the tree in depth-first search. It should be noted that this tree is not balanced. Therefore, the order of tuples inserted into the aggregation tree affects its performance. If the tuples are sorted on the start time and inserted in that order, the aggregation tree would look more like a linked list, causing insertions to be slower than insertions into a balanced binary tree. For this reason, the worst case time to create an aggregation tree is $\mathcal{O}(n^2)$ for n tuples sorted in time. An even more serious limitation of the aggregation tree approach is that the entire tree must be kept in memory. Since the size of an aggregation tree is proportional to the number of distinct timestamps in the temporal relation, the size of the database the aggregation tree algorithm can deal with tends to be limited by the size of available memory and the number of distinct timestamps of tuples.

To minimize memory limitations, a variant of the aggregation tree, called *k-ordered aggregation tree*, was

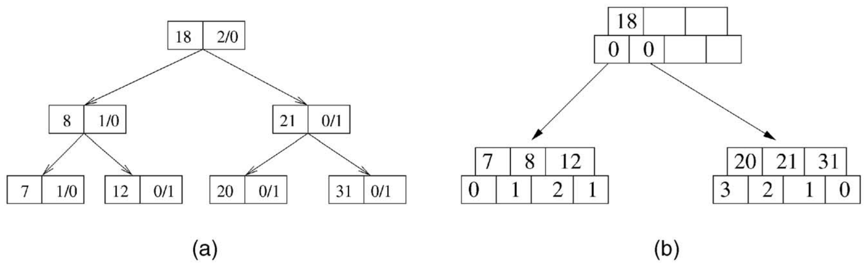


Fig. 1. Nonindexed and indexed temporal aggregation. (a) Balanced tree. (b) SB-tree.

proposed by the same authors. The k-ordered aggregation tree takes advantage of the *k-orderedness* of tuples to enable garbage collection of tree nodes so that the memory requirements can be reduced significantly. However, the k-ordered aggregation tree approach assumes that the tuples in a table are ordered within a certain degree. Specifically, each tuple is at most k positions from its position in a totally ordered version of the table. This requirement is difficult to meet in a real database. Without a priori knowledge about a given table, the k-orderedness is expensive to measure, as it requires an external sort of the table. The worst case running time of the k-ordered aggregation tree algorithm is still $\mathcal{O}(n^2)$.

In an extension of his previous work, Kline [41] proposed using a 2-3 tree, which is a balanced tree, to compute temporal aggregates. The leaf nodes of the tree store the time intervals of the aggregate results. Like the aggregation tree, this approach requires only one database scan. Note that, because it is a balanced tree, the running time is $\mathcal{O}(n \log n)$. However, its main limitation lies in the requirement that a database be initially sorted by start time. It has been shown that, for a randomly ordered database, the aggregation tree performs better than the 2-3 tree approach [41]. This is due to the preprocessing cost required by the 2-3 tree approach to sort the database.

The PA-tree: Kim et al. proposed an algorithm for computing temporal aggregation that is asymptotically better than the aggregation tree. The proposed method is based on the *point-based aggregation tree (PA-tree)* [39], which stores timestamps instead of intervals in an AVL tree. This approach requires one scan of the temporal relation for building the tree. Since the tree is balanced, the time complexity for building the tree is $\mathcal{O}(n \log n)$ rather than $\mathcal{O}(n^2)$ for the aggregation tree. In addition to timestamps, each node in the *PA-tree* stores either a single aggregate value for computational aggregates such as COUNT, SUM, and AVG aggregation, or a list of *value-length* pairs for selective aggregates such as MIN and MAX aggregation. Computing the algebraic aggregate functions is performed by doing an in-order traversal of the tree and updating aggregate values by the amount indicated on each encountered node. Selective aggregate functions are computed by merging the lists of pairs associated to each tree node in similar way to the *skyline problem* [48].

The Balanced Tree: Moon et al. proposed two I/O and computationally efficient algorithms for the evaluation of temporal aggregates [52], [53]. A *balanced tree* is presented for solving temporal aggregation involving computational aggregates (i.e., COUNT, SUM, and AVG). The motivation behind the balanced tree algorithm is that all timestamps in the temporal relation can be sorted *incrementally* by inserting them into a balanced tree as the tuples of an input database are being scanned. Each node of a balanced tree stores a timestamp, either a start time or an end time, and two counters: one storing the number of tuples starting at the timestamp and the other storing the number of tuples ending at the timestamp. Once it has been built, the balanced tree is used to identify constant intervals in the database. Because changes in the database occur only at the timestamps stored in the tree, two consecutive timestamps define a constant interval. The computational cost of processing temporal aggregation is reduced because the query processor needs to generate just one aggregate value per constant interval. This information is sufficient to know what the aggregate value at every point in time is.

Fig. 1a shows the resulting balanced tree for a COUNT aggregate query on the temporal relation given by Table 5a. As a reference, the result for the query is presented in Table 5b. On the balanced tree, processing a query needs an inorder traversal of the tree to find constant intervals. During this traversal, the information on the counters of each node is combined to compute the aggregate value for each constant interval. In the case of a COUNT query, the aggregate value for a constant interval is given by the COUNT of the previous interval plus the number of tuples starting at the start timestamp of the interval minus the number of tuples ending at the start timestamp of the interval.

For queries involving selective aggregates (i.e., MIN and MAX), Moon et al. proposed a *bottom-up* aggregation approach, termed the *merge-sort aggregation* algorithm. Like the classical merge-sort algorithm based on the divide-and-conquer strategy, the merge-sort aggregation algorithm computes a larger (intermediate) aggregate result by merging two smaller (intermediate) aggregate results. The algorithm starts with merging tuples in pairs at the bottom and terminates when a final aggregate result is obtained at the top.

Both of these techniques are constrained by the amount of main memory available in the system. To overcome this limitation, Moon et al. proposed the use of a data structure called the *meta array*. By using the meta array, tuples in the base relation can be grouped into small subsets (following some partition of the time line) for which temporal aggregation can be computed given a limited buffer space. The meta array will maintain the aggregate information of tuples overlapping the intervals given by this time partition to guarantee the correctness of the result.

Parallel Temporal Aggregation: Here, we discuss algorithms that have been developed for the parallel processing of temporal aggregation in large-scale databases. Ye and Keane proposed two approaches to parallelize the aggregation tree algorithm on a shared-memory architecture [85]. They propose parallelizing temporal aggregation queries

TABLE 5
T-Employees: A Sample Temporal Relation:
(a) Temporal Relation
(b) Its COUNT Temporal Aggregation

Name	Salary	Begin	End
Frank	46,000	18	31
Praveen	45,000	8	20
Sophia	35,000	7	12
Sophia	38,000	18	21

(a)

COUNT	Begin	End
1	7	8
2	8	12
1	12	18
3	18	20
2	20	21
1	21	31

(b)

that include GROUP-BY on explicit attributes. Each group defined by the grouping attribute is send to a processor where the temporal aggregation is computed locally.

Gendrano et al. have also developed several parallel algorithms [25] for computing temporal aggregates, specifically on a shared-nothing architecture, by parallelizing the aggregation tree algorithm. Gendrano et al. showed promising scale-up performance of the parallel algorithms through extensive empirical studies under various conditions. Nonetheless, all the aforementioned parallel algorithms inherit the same limitations from the aggregation tree algorithm as the parallel algorithms were developed by parallelizing the aggregation tree. In particular, the size of the database those parallel algorithms can handle will be limited by the aggregate memory of participating processors.

Moon et al. [52], [53] extended the notion of meta array to cover several processors while computing temporal aggregates in parallel. A global meta array maintains aggregate information about tuples overlapping the time interval assigned to each processing node, whereas local meta arrays are used to compute temporal aggregation locally on each node.

All the nonindexed evaluation algorithms for temporal aggregation presented here address the same type of query. At a logical level, this type of query can be described as follows: First, for each time value τ at the finest granularity supported by the temporal relation, a collection of tuples is generated. The collection corresponding to the time value τ is formed by all tuples in the temporal relation valid during time τ. Second, an aggregate value is generated for each collection and the corresponding time value τ is annotated with this aggregate value. Finally, consecutive time values annotated with the same aggregate value are coalesced into a constant interval, that is, a time interval for which the temporal aggregate value remains constant. Note that this type of query corresponds to temporal aggregation queries using temporal group composition as presented by Definition 3. The granularity used during group composition equals the finest granularity supported by the temporal relation.

4.2.2 Indexed Aggregation Evaluation

A more recent approach for evaluating temporal aggregation queries was proposed by Yang et al. They introduced the *SB-tree* [84] for incrementally computing temporal aggregates using a materialized view approach. The SB-tree was developed for a data warehouse environment in which mostly insertions are expected. If deletion operations are expected, then MIN and MAX aggregation queries are not supported since these aggregate values cannot be incrementally maintained under deletions. The SB-tree is a disk-based structure that combines aspects of the segment tree and the B-tree. It creates a balanced tree just as the B-tree and, at the same time, it maintains a hierarchy of intervals. Each of these intervals is associated with a partially computed aggregate. Fig. 1b shows the resulting COUNT SB-tree for Table 5a. Aggregation over a given temporal interval is evaluated by performing a depth-first search on the tree and accumulating the partial aggregate values along the way. For instance, generating the aggregate value for the interval $(7, 8)$ in Fig. 1b requires visiting the root node. The root node indicates that, for interval $(7, 8)$, we need to accumulate the preaggregate value 0. When the leaf node is visited, it indicates that the value 1 needs to be added to the partial aggregate 0. Therefore, the aggregate value for the interval $(7, 8)$ is 1.

In addition to supporting temporal queries involving temporal group composition, the SB-tree supports queries that require a sliding window termed in their paper *cumulative aggregate queries*. For every time value τ at the finer granularity supported by the temporal relation, a cumulative aggregate query defines a window frame around τ using a time interval of length w preceding τ. All tuples valid during the interval $[\tau - w, \tau]$ form a collection for which an aggregate value is generated. Cumulative aggregate queries can be defined by our model using temporal partitioning and sliding windows. The granularity used for the sliding window definition should be the finest granularity supported by the temporal relation. The granularity used for the temporal partition definition should be so coarse that all tuples in the temporal relation belong to the same collection.

One drawback of the SB-tree lies in the assumption that aggregate queries are always evaluated over the entire base relation. This is a clear disadvantage because aggregate queries usually specify a number of predicates to select the tuples on which temporal aggregation should be computed. The *multiversion SB-tree* (MVSB-tree) introduced by Zhang et al. [89] was specifically designed to address this issue. It was proposed to deal with temporal aggregate queries coupled with range predicates on explicit attributes, termed *range temporal aggregates* [89]. The MVSB-tree is logically a series of SB-trees, one for each timestamp. Given a range on the values of one of the explicit attributes r and a temporal interval i, the MSVB-tree computes the aggregate of all the tuples within r and valid during i as a series of additions and subtractions of values stored in the index. Because this is a form of preaggregation, only distributive aggregate functions can be evaluated by the MVSB-tree, in particular SUM, COUNT, and AVG.

The effectiveness of the MVSB-tree is limited by the size of its index, which can be larger than the database [74]. This limitation was overcome by Tao et al. [74] using an approach that computes an approximate solution to aggregate queries while maintaining only a small index. This approach is based, at a logical level, on a MVB-tree, but can be practically implemented using an B-tree and an R-tree. In particular, Tao et al. can approximately evaluate queries containing COUNT and SUM aggregate functions. The approaches presented by Zhang et al. [89] and Tao et al. [74] were designed for the evaluation of nonsequenced queries (i.e., the query does not result in a temporal relation). Once a set of tuples has been selected by a range predicate given on one of the explicit attributes and by a temporal predicate, the temporal information of the tuples is discarded. Therefore, the time-evolving information of the aggregate value for the time interval indicated by the query is lost. We should note that, while sequenced aggregate queries can be evaluated using these approaches by issuing a query for every time value at the desired granularity, such an evaluation might not be efficient. In addition, there is a side effect of ignoring the temporal nature of data known as the *distinct count problem*. This problem occurs when a temporal object remains in the query range for several timestamps during the query time interval. In such a case, the same object could be counted multiple times [72], unless the algorithm explicitly avoids doing so.

One drawback of indexed evaluation algorithms is that they either assume that aggregate queries do not include predicates on explicit attributes [84] or they assume that predicates are defined on a single attribute [74], [89]. Another disadvantage of these algorithms is that they can only process certain types of aggregate functions. In particular, partial aggregation, or preaggregation, can only be used for distributive functions such as those included in the SQL-92 standard [29]. *Holistic* aggregate functions (e.g., MEDIAN) cannot be combined with preaggregation. Therefore, queries involving holistic aggregate functions cannot be processed with indexed evaluation methods.

4.3 Aggregates on Data Streams

Data streams are ordered sequences of value points that are read/received in increasing order [4]. Because each value in a data stream is usually associated with a timestamp indicating either the time when the value was generated or the time when the value was received, data streams may be considered a special case of temporal data. Applications requiring the use of data streams are increasingly common and it is easy to find examples of data streams applications, such as network monitoring, security, telecommunications data management, web applications, manufacturing, and sensor networks [4], [17], [88].

Because of the immense amount of data generated by the stream, it is extremely costly to store all data in such a way that it is readily available for answering queries. Instead, stream data is either discarded or archived after having been *looked at* just once. In consequence, most applications only perform aggregate queries over data streams. Summarized information about the data stream is often more important than retrieving specific entries with certain properties [62].

There are two models used for processing stream data [16], [88]. The *sliding window model* is used when only recent values in the data are of interest (i.e., within the past w timestamps). The *complete (or infinite window) model* is used when all values in the stream are of interest. While stream data is a special case of temporal data, the complete model essentially ignores its temporal properties.

The sliding window model for processing stream data corresponds to temporal partition composition and temporal sliding window composition in our model. The sliding window composition is performed at the finest granularity supported by the timestamps on the values of the stream. The window frame is given by a trailing temporal interval of size w and there is no leading temporal interval. The temporal partition composition is such that the entire stream data creates only one collection of values. Methods developed to compute aggregation over data streams using the sliding window model include those by Datar et al. [16] and Zhang et al. [88]. Datar et al. present a method for computing approximate solutions for the COUNT and SUM aggregate functions. For this, they propose the use of *Exponential Histograms*, a data structure that can be incrementally maintained while preserving guarantees on the approximate solutions to COUNT and SUM aggregate queries. Zhang et al. [88] propose a mechanism for computing temporal aggregation on stream data based on a hierarchy of granularities. The main idea is to use different granularities to aggregate data depending on its age. Older data is aggregated at a coarser granularity, whereas the most recent data is aggregate at the finest granularity. The most recent data is aggregated following the sliding window model. Established systems for stream data management have also adopted this approach for computing aggregate functions. One good example is Aurora [1], which is a model and an architecture for data stream management.

The complete model for processing data streams considers all values in the stream read so far. Since data is not available at query time, only approximate solutions to aggregate queries are possible. For this, research work has turned to the maintenance of summarized information in the form of histograms [30], [62] or sketches [15], [17]. We do not provide further details on these methods because they ignore the temporal characteristics of data while evaluating aggregate queries.

5 Spatial Aggregation

Spatial data appear in numerous applications, such as GIS, multimedia, and even traditional databases. Spatial database systems organize and manage large amounts of multidimensional data. Objects stored in spatial relations are associated with spatial extents that define their geometric features [47]. These objects are usually points, lines, polygons, and volumetric objects [81]. Spatial relations are indexed by multidimensional access methods, such as R-trees, for the efficient processing of queries such as spatial selections or spatial joins [24], [47], [66]. Due to the complexity of the spatial operators, the large amount of data, and the difficulty in defining a spatial ordering, a traditional relational Database Management System (DBMS) may not be adequate to efficiently support spatial data. Therefore, DBMSs must offer spatial query processing capabilities to meet the needs of such applications [6], [47], [49], [81].

Aggregate queries over spatial data require the organization of tuples from a spatial relation into collections based on their spatial extent. Aggregate functions are then applied to these collections. Spatial aggregation, as studied in the literature, can be viewed as aggregates on collections of tuples based on granularities of the spatial domain. A spatial domain may be represented as a set (e.g., R^3, R^2, N^3, N^2), with elements referred to as points. However, for geographic applications, horizontal space (e.g., latitude and longitude) is usually segregated from vertical space (e.g., depth or altitude), with horizontal and vertical granularities defined on the spatial domain [38].

A horizontal spatial granularity may be defined as a mapping from the integers to a subset of the space domain such that 1) granules from a spatial granularity do not overlap and 2) the index set of a spatial granularity provides a contiguous encoding. Different granularity levels for a horizontal space could be expressed in *degree*, *minute*, or *second*, for example. The definition associated with vertical spatial granularity is similar to temporal granularity. Different levels of granularity for the vertical space could be expressed using *centimeter*, *meter*, and *kilometer*, for example. A three-dimensional granularity is a cross product of the horizontal and vertical spatial granularities [38].

5.1 Formal Definition of Spatial Aggregation

Different levels of spatial granularities can be used to define group, partition, and sliding window composition in a spatial relation. In *spatial group composition*, tuples sharing the same space value at granularity G form a collection termed *group*. For each group, an aggregate function is applied and the group is annotated with the aggregate value. When computing spatial aggregation using group composition, the resulting relation is a spatial relation defined at granularity G (i.e., the granularity of the groups). Consider, for example, a land management application that keeps track of forests in the US. The regions of land covered by forest can be estimated from satellite data such as Landsat [43]. The following is an example of spatial aggregation query for this application.

Query 5. *Compute the amount of land covered by forest in every county of the state of Arizona.*

This query can be answered by applying group composition at granularity $G = county$ to the tuples that satisfy the predicate "in the state of Arizona." Then, for each collection of tuples sharing the same spatial value (i.e., same county), an aggregate function is applied. In this case, we apply `SUM` on the *area*, where area is a property of any object with a spatial extent. The answer to this kind of query can be formally defined as follows:

Definition 5 (Aggregation Using Spatial Group Composition). *Given a spatial relation R^S and a select predicate SP, let $\mathcal{S}(G, R^S) = \{s | s \in cast(r[se], G) \wedge r \in R^S\}$ be the spatial counterpart of $\mathcal{T}(G, R^T)$, where $r[se]$ gives the spatial extent of tuple r. Each space value $s \in \mathcal{S}$, defines a subset of tuples of R^S based on a space partition as follows:*

$$P_{G,SP}(s, R^S) = \{t \mid \exists\, r \in R^S \wedge overlaps(cast(r[se], G), s) \wedge SP(r) \wedge \ t[A_1 \dots A_n] = r[A_1 \dots A_n] \wedge \ t[se] = intersect(r[se], s)\}. \tag{3}$$

The result of an aggregate query using spatial group composition at granularity G is given by the following expression:

$$GBAgg_{f_i,G,SP}(R^S) = \{s \circ f_i(P_{G,SP}(s, R^S)) | s \in \mathcal{S}(G, R^S)\}.$$

In (3), $r[A_1 \dots A_n]$ indicates the explicit attributes of tuple $r \in R^S$, whereas $r[se]$ indicates its implicit spatial extent. Depending on the type of this spatial extent, (3) may or may not define a strict partition of the data. If the base spatial relation stores only point objects, a partition of the spatial domain also defines a partition of the data. On the other hand, if the objects stored in the relation correspond to regions (say areas with different vegetation), then (3) defines a subset of the rows in R^S rather than a strict partition. Similar to the case of (2), the spatial extent of the tuples in P has been narrowed to a single granule of the spatial granularity G.

Spatial partition composition is used when a finer level of aggregation is required. During this process, each space value at granularity H ($H \prec G$, where G is the granularity used for spatial group composition) defines a collection of tuples termed a partition. To each partition, we apply *spatial sliding window composition*, which places a window frame around each spatial value s at granularity J ($J \prec H$). A window frame around $s =< s_x, s_y >$ is defined as

$$W_{windowsize}(s) = \{< x, y > | (s_x - windowsize \leq x \leq s_x + windowsize) \wedge (s_y - windowsize \leq y \leq s_y + windowsize)\}, \tag{4}$$

where *windowsize* is a query argument defining the size of the window frame. In this case, s was a two-dimensional spatial point and the window frame was a square. However, (4) can be generalized for three-dimensional spaces and for different shapes of windows. For every space value s, an aggregate value is generated by applying an aggregate function to the set of tuples valid for the window defined around s.

When computing spatial aggregation using partition and sliding window composition, the resulting relation is a spatial relation containing one entry for every spatial value at granularity J (i.e., the granularity used for sliding window composition). To illustrate this, consider the land management application described before, from which we would like to detect a good place for founding a natural reserve. In this case, we are interested in analyzing information at a fine granularity. Clearly, finding the county with the highest plant diversity is of not much use in this case. The result of the following query might provide the required information.

Query 6. *Compute the average diversity of vegetation (i.e., number of species of plants) per square kilometer in each county of the state of Arizona. For each square kilometer, consider neighboring regions up to 2 km on each direction (north, south, east, and west) to smooth out local variations.*

To answer this query, we need spatial partition composition using a granularity at the county level. Within each partition, we define sliding window composition using a granule of 1 km^2. The aggregate function is then applied to the set of tuples occurring within the limits of each spatial window and that satisfy the spatial predicate "in the state of Arizona." In this case, the window frame is 5×5 km because leading or trailing spatial intervals of 2 kilometers are used to account for the influence that neighboring regions might have on the diversity of a particular region. In general, the answer for this type of queries can be formally expressed by the following definition.

Definition 6 (Aggregation Using Spatial Partition Composition). *Given a spatial relation R^S and a select predicate SP, let us use $\mathcal{S}(H, R^S)$ and $P_{H,SP}(s, R^S)$ as before. Let J be a*

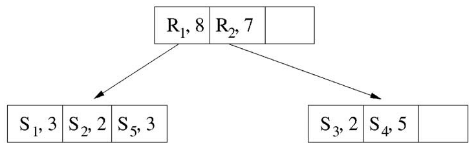


Fig. 2. The aR-tree for `COUNT` aggregation of Table 6.

space granularity with $J \prec H$. For each $s \in \mathcal{S}(J, R^S)$, a window frame with respect to the spatial partition generated by granularity H is defined as

$$WF_{H,SP,J,windowsize}(s, R^S) = \{r | r \in P_{H,SP}(cast(s, H), R^S) \wedge overlaps(cast(r[se], J), W_{windowsize}(s))\},$$

where windowsize is a query argument that defines the window frame around each space value s within a window partition.

The result of an aggregate query on spatial relation R^S with spatial partitions at granularity H and window frames at granularity J with range defined by windowsize is given by

$$WAgg_{f_i,H,J,windowsize}(R^S) = \{cast(s, H) \circ s \circ f_i(WF_{H,J,windowsize}(s, R^S)) | s \in \mathcal{S}(J, R^S)\}.$$

5.2 Existing Approaches for Evaluating Spatial Aggregate Queries

Various algorithms have been proposed to evaluate aggregate queries on spatial databases. Because these queries usually include a spatial selection predicate describing a multidimensional box or window, they are often referred to as *box aggregation* queries. Aggregate queries with these spatial predicates retrieve summarized information of the objects that either partially or completely overlap the region defined by the multidimensional window [45], [75], [81], [91].

In Section 4, we classified existing approaches for the evaluation of temporal aggregates into either nonindexed or indexed evaluation algorithms. However, for spatial aggregation, we have only encountered indexed evaluation algorithms. That is, rather than answering directly from the data stored in a spatial relation, they rely on a small disk-based data structure to answer the queries. Furthermore, to the best of our knowledge, all these methods only focus on box aggregation queries.

Pedersen and Tryfona [59] proposed preaggregation over spatial data warehouses. They analyze the properties of topological relationships between two-dimensional spatial objects and show why traditional techniques for preaggregation will not work on these settings. Preaggregation is a common technique used to efficiently process aggregate functions over data warehouses. However, for preaggregation to work, the spatial properties of the objects must be distributive over some aggregate function. On spatial data, some of the topological relationships are not distributive (e.g., union). To circumvent this problem, Pedersen and Tryfona presented a methodology to decompose spatial objects in such a way that preaggregation can be applied.

Using an approach that combines indexing with preaggregation, Papadias et al. presented the *Aggregation R-tree (aR-tree)* [56], an R-tree that annotates each MBR with the value of the aggregate function for all the objects that are

TABLE 6
A Sample Spatial Relation

Road Segment	Car Count	x_1	y_1	x_2	y_2
S_1	3	1	1	5	5
S_2	2	2	8	10	8
S_3	2	21	11	26	20
S_4	5	28	21	28	11
S_5	3	8	6	11	1

enclosed by it. Therefore, an aggregate query does not need to access all the enclosed objects since part of the answer is found in the intermediate nodes of the tree. In this case, preaggregation is possible because they only consider disjoint spatial objects. We show an example of the aR-tree in Fig. 2. The example shown in this figure represents an aR-tree built for the `COUNT` aggregation of the spatial relation given in Table 6. This table is similar to the examples used by Papadias et al. [56]. It represents road segments for which we keep track of the amount of traffic. That is, we count the number of vehicles per road segment. In Table 6, the spatial attributes of the road segments are their start and end coordinates (x_1, y_1) and (x_2, y_2), respectively. The aR-tree has one entry for each road segment and each entry is annotated with the corresponding vehicle count. To answer a query, only nodes intersecting the spatial range given by the query are traversed. Index nodes that are fully contained by the query range are not traversed. Instead, only the aggregate information of the node is used. This process is recursive and it stops once a leaf node is reached or no index node overlaps the query range.

Zhang and Tsotras presented a set of four optimization techniques to improve query performance for `MIN` and `MAX` aggregation [90]. While some of these optimizations could be implemented in the aR-tree, they also proposed the *Max R-tree (MR-tree)*, a data structure explicitly designed to maintain `MIN` and `MAX` aggregates. In later work, Zhang et al. [91] focus on developing efficient solutions to the `COUNT`, `SUM`, and `AVG` aggregate functions. Instead of relying on previous indexing techniques such as the aR-tree, they use specialized aggregate indexes that incrementally maintain aggregates. They provide a new approach to reduce aggregate queries to the *dominance-sums* problem. In addition, they extend the best known solution to the *dominance-sums* problem, the *ECDF-tree* [5], a static, main-memory data structure, and make it dynamic and disk-based (the *ECDF-B-tree*). Unfortunately, this data structure cannot efficiently handle insertions when optimized for queries. They present a solution to this problem by introducing the *Box Aggregation Tree* (BA-tree), a data structure that efficiently supports both insertions and queries.

Lazaridis and Mehrotra proposed a tree structure for evaluating box aggregate queries in a multidimensional space containing point data items [45]. Their approach uses a tree structure called *Multiresolution Aggregate tree (MRA-tree)* and their algorithm selectively traverses nodes of this tree based on reasonable assumptions on which nodes, if examined, will most likely reduce the uncertainty on the value of the aggregate. Tree nodes are augmented with aggregate information for all data points indexed by them. Tao et al. [75] use a specialized index structure called the *aggregate Point-tree (aP-tree)* for evaluating box aggregation queries over points in two-dimensional space. The intuition behind the aP-tree is that two-dimensional points can be viewed as intervals in the key-time plane and, therefore, they can be indexed using temporal access methods. A box aggregation query is reduced to a pair of *vertical range aggregate* (VRA) queries, which can be answered in constant time by the aP-tree. The main advantage of this approach is that the query cost is independent of the number of objects contained by the query window.

6 Spatiotemporal Aggregation

An increasing number of applications manage spatiotemporal aspects of the real-world. In consequence, we have observed a growing interest for this kind of application in the research community. Recent surveys and bibliographic studies show a large amount of research papers on spatiotemporal databases [2], [63]. A spatiotemporal object is an object with both spatial and temporal extent [12], [82]. Not only the spatial extents of these objects can change over time, but also the values of the explicit attributes describing nonspatial characteristics of the object may change over time [35], [78].

There are various approaches to modeling the time-varying spatial properties of spatiotemporal objects. Some of them consider objects which observe continuous movement [13], [60], [61], while others consider objects that change its shape in discrete steps [49], [76], [77]. However, these modeling approaches only affect how data should be stored and organized. From a semantic perspective, the time model adopted is largely irrelevant for the computation of aggregate functions.

Aggregate queries over spatiotemporal data require the organization of tuples from a spatiotemporal relation into collections defined based on their spatial and temporal extents. Aggregate functions are applied to these collections. In Sections 4 and 5, we have shown how collections of tuples are generated based on granularities of the temporal and spatial domains, respectively. This concept can be extended to generate collections of tuples based on spatiotemporal granularities. A spatiotemporal granularity is a cross product of the spatial and temporal granularities.

6.1 Formal Definition of Spatiotemporal Aggregation

Different levels of spatiotemporal granularities can be used to define group, partition, and sliding window composition in a spatiotemporal relation. In *spatiotemporal group composition*, tuples sharing the same spatial and temporal value at granularity G form a collection termed *group*. For each group, an aggregate function is applied and the group is annotated with the aggregate value. When computing spatiotemporal aggregation using group composition, the resulting relation is a spatiotemporal relation defined at granularity $G = G^S \times G^T$ (i.e., the cross product of spatial granularity G^S and temporal granularity G^T).

Consider a land management application that keeps track of the forests in the US. In this application, each stored object is a region (spatial extent) that can change with time (temporal extent). Forests can change their shape due to natural phenomena such as wildfires or droughts. In addition, forest composition is also time-varying because vegetation changes with the seasons. Information about spatiotemporal changes in the forest can be obtained by remote sensors such as the *Moderate Resolution Imaging Radio Spectroradiometer (MODIS)* [37], [64] aboard the Terra and Aqua Satellites. This information is available for temporal granularities as fine as 16 days and spatial granularities of 500 meters [51]. For land management applications, we might be interested in identifying correlations between wildfires and forest density. A useful query in this case will be the following:

Query 7. *For every county in the state of Arizona, what has been the yearly forest density for the last 10 years?*

In this case, we are not only interested in knowing the aggregate value "density," but also we want to know how this value changes in time and space. Note that Query 7 refers to the spatiotemporal granularity $G = county \times year$. The clause "for the last 10 years" is a temporal predicate. Similarly, the clause "in the state of Arizona" is a spatial predicate. These predicates are used to select qualifying tuples. To answer this query, we need to form groups of tuples sharing the spatial value $county$ and the temporal value $year$. Each of these groups is then annotated with their corresponding aggregate value.

Some queries may require aggregate at a finer level of detail while still maintaining some data organization at a higher level. In such a case, *spatiotemporal partition composition* and *spatiotemporal sliding window composition* are required. When computing spatiotemporal aggregation using partition and sliding window composition, the resulting relation is a spatiotemporal relation containing one entry for every pair of spatial and temporal values at granularity J (i.e., the granularity used for sliding window composition). However, each window frame can only contain data valid during a temporal value at granularity H (the granularity used for partition composition), $J \prec H$.

To illustrate the need for spatiotemporal partition and sliding window composition, consider the following scenario. During year 2002, the state of Arizona experienced some of the worst wild fires in its history. By identifying characteristics of the vegetation (say density) that could have influenced these wild fires, we might be able to prevent similar wild fires and minimize ecological and property damage. What is needed is a fine-grained analysis of the forest properties such that particular places (say lodging cabins) can be evacuated or measures taken to prevent wildfires. At the same time, we want to perform this analysis within human-defined spatial boundaries (say a county) to notify the proper authorities. At the same time, we need to know how the vegetation changes over time. In addition, for every region under analysis, we might want to consider the conditions of neighboring regions (i.e., it does not help to clean a particular property within an acre of land free of trees if it is surrounded by a dense forest; the property is still likely to suffer fire damage). For this application, the following query might provide useful information.

Query 8. *For every square kilometer in every county in the state of Arizona, what has been the density of the forest for this year? For each square kilometer, consider neighboring regions up to 2 km on each direction (north, south, east, and west). For obtaining the density, consider the current and next oldest MODIS observation.*

This query is expressed using two granularities, $H = county \times year$ and $J = km^2 \times (16\text{-}day)$. The clauses "in the state of Arizona" and "for this year" are spatial and temporal predicates for selecting qualifying tuples. To answer Query 8, we need spatiotemporal partition composition using a spatial granularity at the county level and temporal granularity in years (however, only one year is of interest). Within each partition, we define spatiotemporal sliding window composition using spatial granules of 1 square kilometer and temporal granules of 16 days (i.e., the finest temporal granularity in the data set). The aggregate function is applied to the set of tuples occurring within the limits of each spatiotemporal window. In this case, the window is 5×5 km by two 16-day temporal granules (i.e., current and previous).

6.2 Existing Approaches for Evaluating Spatiotemporal Aggregate Queries

A typical spatiotemporal query specifies spatial and temporal predicates to select tuples of interest. A spatial predicate is defined in terms of a point or an extent, while a temporal predicate can involve a time instant or a time interval [49], [73]. Algorithms proposed for the evaluation of spatiotemporal aggregation queries seem to concentrate in a generalization of the box aggregation problem presented in Section 5. In this case, the query box is extended with a temporal interval. Given a spatial range and a temporal interval, the query returns summarized information of all the tuples valid during the time interval and that are contained or intersected by the query range. Alas, the evaluation of this type of queries does not result in a spatiotemporal relation.

The evaluation of spatiotemporal aggregation queries has only recently caught the attention of the research community. Here, we present, in chronological order, four different approaches for the evaluation of these queries proposed in the last three years. All these methods are based on a disk-based data structure that stores some precomputed values that are used for answering the queries. Therefore, they all are indexed evaluation algorithms.

Zhang et al. [89], [87] proposed an extension to their approach for computing aggregates over data streams to handle spatiotemporal data. This approach, based on multiple granularity levels was described in Section 4 and we omit further details here. Papadias et al. [57] proposed an index-based approach in which they group spatial objects into static regions and index these regions using an R-tree. Each region represented in the R-tree is annotated with aggregate information over all the timestamps in the base relation. In addition, for every region represented in the R-tree, the proposed data structure maintains a B-tree that contains time-varying aggregate data about the region. This data structure is called the *aggregation RB-tree* (aRB-tree) and can be extended to handle the case when objects are grouped into dynamic regions, resulting in the *aggregation Historical RB-tree* (aHRB-tree) or the *aggregate three-dimensional R-B-tree* (a3DRB-tree).

One drawback of Papadias et al.'s aRB-tree is the *distinct counting problem*. This problem occurs if a data object remains in the query region for several timestamps during the query interval because such data object will be counted multiple times [72]. Tao et al. [72] recognize the distinct count problem within the aRB-tree and presented an approximate approach to evaluating distinct COUNT and distinct SUM aggregate queries. Their approach is based on sketches and an sketch index, similar in structure to the aRB-tree.

Another approach for the approximate evaluation of spatiotemporal aggregate queries was proposed by Sun et al. [71]. They consider a data model in which moving objects continuously generate large amounts of spatiotemporal information in the form of data streams. Until an object transmits a new location, it is assumed to be in the last recorded position. Space is partitioned in a two-dimensional grid of $w \times w$ regular cells, where w is a constant called resolution. Each cell is associated with the number of objects (at present time) in its extent. Sun et al.'s approach can answer approximate queries to the COUNT aggregate based on a data structure termed *Adaptive Multidimensional Histogram (AMH)*, which is updated every time an object transmits a new position. The AMH can only be used for answering snapshots aggregate queries [71].

7 Research Opportunities

We have observed that the proposed algorithms for the evaluation of temporal, spatial, and spatiotemporal aggregation rely on some form of preaggregation. That is, they precompute results for a particular set of qualifying tuples and keep the summarized information in a hierarchical data structure. To evaluate a query, the data structure is traversed and the values found at different levels of the hierarchy are combined to obtain the query result. While these approaches are efficient, they can only answer aggregate queries for a specific set of tuples. It is not clear whether these approaches can be easily modified to handle aggregate queries with arbitrary predicates. Different predicates will select different set of tuples. rendering previous precomputed values useless. We consider extending these approaches for handling arbitrary predicates an interesting research problem.

Another research opportunity we have identified has its basis in the fact that only a small number of the proposed approaches were designed for the evaluation of *sequenced* aggregate queries (i.e., the query result is a relation of the same nature as the base relation). This opportunity is more evident for spatial and spatiotemporal databases. For these cases, the proposed approaches focus on the box aggregation problem. After applying spatial and temporal predicates for selecting tuples, the spatial and temporal properties of the qualifying tuples are ignored. We should note that we can still evaluate sequenced aggregate queries using box aggregation. For instance, we could issue a box query for every time and space value at the desired granularity, then combine the individual results. However, such an approach would not be very efficient. In addition, problems such as the distinct count problem mentioned by Tao et al. [72] would have not arisen if the temporal properties of the selected tuples had been preserved and considered while computing the aggregation.

When comparing the proposed approaches to our model, we realize the need for algorithms that evaluate spatiotemporal aggregate queries performing spatiotemporal group and partition composition. Current approaches for evaluating spatiotemporal aggregation do not offer support for this kind of queries. Queries such as Query 7 and Query 8, for example, cannot be evaluated by the surveyed techniques. Nor can these techniques evaluate Queries 5 and 6.

As we have mentioned before, the existing approaches for the efficient evaluation of temporal, spatial, and spatiotemporal aggregate queries rely on preaggregation. Preaggregation is not useful if the aggregate functions in the query are nondistributive (e.g., MEDIAN); we need to develop algorithms for the efficient evaluation of this type of aggregate functions for spatiotemporal data.

8 Conclusion

In this paper, we have studied the most relevant techniques for the evaluation of aggregate queries on spatial, temporal, and spatiotemporal data. We have also presented a model that reduces the evaluation of aggregate queries to the problem of selecting qualifying tuples and grouping these tuples into collections on which an aggregate function is to be applied. This model gives us a framework that allows us to analyze and compare the different existing techniques for the evaluation of aggregate queries. At the same time, it allows us to identify opportunities of research on types of aggregate queries that have not been studied.

Algorithms for the evaluation of aggregate queries can be classified as either nonindexed or indexed. Nonindexed algorithms need to scan the base relation every time the query is issued. During this scan, aggregate values are incrementally computed. Indexed algorithms, on the other hand, rely on annotated disk-based data structures. These structures provide sufficient information for computing aggregates while not requiring the evaluation algorithm to explore every qualifying object in the base relation.

As we have indicated, most of the existing approaches for the evaluation of temporal, spatial, and spatiotemporal aggregate queries rely on some form of preaggregation. Hence, they only consider distributive aggregate functions such as COUNT, SUM, and MAX. Efficient methods for computing nondistributive aggregate functions such as MEDIAN, MODE, or RANK should be proposed. This issue has recently been addressed for traditional databases by Palpanas et al. [55]. They propose a general incremental maintenance mechanism that applies to all aggregate functions.

We note that *sequenced* aggregate queries have not been addressed for spatial and spatiotemporal databases. A sequenced temporal query is one that is effectively evaluated at every granule in time, resulting in a temporal relation [68]. Sequenced temporal aggregation can be evaluated using Definitions 3 and 4. We can extend this term and define a sequenced query as one resulting in a relation of the same type as the base relation. Sequenced spatial aggregation can be evaluated using Definitions 5 and 6. It is important that we evaluate an aggregate function without ignoring the spatial and temporal characteristics of the data. This is a critical issue because it is important that we know both the spatial and temporal properties of aggregate values. For example, consider a weather monitoring application keeping track of a hurricane. In such applications, it is not only of interest to know the maximum speed of the wind, but also when and where such strong wind is expected.

Acknowledgments

This work was sponsored in part by US National Science Foundation (NSF) Grant No. IIS-0100436 and NSF Research Infrastructure Program EIA-0080123. It was also supported in part by the Mexican Foundation for Science and Technology (CONACyT), scholarship 117476. The authors assume all responsibility for the contents of the paper.

References

[1] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A New Model and Architecture for Data Stream Management," *VLDB J.,* vol. 12, no. 2, pp. 120-139, Aug. 2003.

[2] T. Abraham and J.F. Roddick, "Survey of Spatio-Temporal Databases," *GeoInformatica,* vol. 3, no. 1, pp. 61-99, 1999.

[3] S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi, "On the Computation of Multidimensional Aggregates," *Proc. VLDB Conf.,* pp. 506-521, Sept. 1996.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems ," *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems,* pp. 1-16, June 2002.

[5] J.L. Bentley, "Multidimensional Divide-and-Conquer," *Comm. ACM,* vol. 23, no. 4, pp. 214-229, 1980.

[6] E. Bertino, B.C. Ooi, R. Sacks Davis, K.-L. Tan, J. Zobel, B. Shidlovsky, and B. Catina, *Indexing Techniques for Advanced Database Systems.* Boston: Kluwer Academic, 1997.

[7] C. Bettini, C.E. Dyreson, W.S. Evans, R.T. Snodgrass, and X.S. Wang, "A Glossary of Time Granularity Concepts," *Temporal Databases: Research and Practice,* pp. 406-413, Springer, 1998.

[8] C. Bettini, S. Jajodia, and S.X. Wang, *Time Granularities in Databases, Data Mining, and Temporal Reasoning.* Berlin: Springer, 2000.

[9] L. Cabibbo and R. Torlone, "A Framework for the Investigation of Aggregate Functions in Database Queries," *Proc. Int'l Conf. Database Theory (ICDT),* pp. 383-397, Jan. 1999.

[10] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayva, "Overcoming Limitations of Sampling for Aggregation Queries," *Proc. Int'l Conf. Data Eng.,* pp. 534-542, Apr. 2001.

[11] S. Chaudhuri, G. Das, and V. Narasayva, "A Robust, Optimization-Based Approach for Approximate Answering of Aggregate Queries," *Proc. ACM-SIGMOD Conf.,* pp. 295-306, May 2001.

[12] C.X. Chen and C. Zaniolo, "SQL^{ST}: A Spatio-Temporal Data Model and Query Language," *Proc. Int'l Conf. Conceptual Modeling (ER),* pp. 96-111, Oct. 2000.

[13] Y.-J. Choi and C.-W. Chung, "Selectivity Estimation in Spatio-Temporal Queries to Moving Objects," *Proc. ACM-SIGMOD Conf.,* pp. 440-451, June 2002.

[14] S.-J. Chun, C.-W. Chung, J.-H. Lee, and S.-L. Lee, "Dynamic Update Cube for Range-Sum Queries," *Proc. VLDB Conf.,* pp. 521-530, Sept. 2001.

[15] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate Aggregation Techniques for Sensor Databases," *Proc. Int'l Conf. Data Eng.,* pp. 449-460, 2004.

[16] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining Stream Statistics over Sliding Windows (Extended Abstract)," *Proc. Ann. ACM-SIAM Symp. Discrete Algorithms,* pp. 635-644, Jan. 2002.

[17] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi, "Processing Complex Aggregate Queries over Data Streams," *Proc. ACM-SIGMOD Conf.,* pp. 61-72, June 2002.

[18] C.E. Dyreson, W.S. Evans, H. Lin, and R.T. Snodgrass, "Efficiently Supporting Temporal Granularities," *IEEE Trans. Knowledge and Data Eng.,* vol. 12, no. 4, pp. 565-587, July/Aug. 2000.

[19] C.E. Dyreson, F. Grandi, W. Kafer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J.F. Roddick, N.L. Sarda, M.R. Scalas, A. Segev, R.T. Snodgrass, M.D. Soo, A. Tansel, P. Tiberio, G. Wiederhold, and C.S. Jensen, "A Consensus Glossary of Temporal Database Concepts," *SIGMOD Record,* vol. 23, no. 1, pp. 52-64, Mar. 1994

[20] R. Epstein, "Techniques for Processing of Aggregates in Relational Database Systems," Technical Report UCB/ERL M7918, Univ. of California, Berkeley, Feb. 1979.

[21] M. Erwing, R.H. Guting, M. Schneider, and M. Vazirgiannis, "Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases," *GeoInformatica,* vol. 3, no. 3, pp. 269-296, 1999.

[22] L. Forlizzi, R.H. Guting, E. Nardelli, and M. Schneider, "A Data Model and Data Structures for Moving Object Databases," *Proc. ACM-SIGMOD Conf.*, pp. 319-330, May 2000.

[23] J.C. Freytag and N. Goodman, "Translating Aggregate Queries into Iterative Programs," *Proc. VLDB Conf.*, pp. 138-146, Aug. 1986.

[24] V. Gaede and O. Gunther, "Multidimensional Access Methods," *ACM Computing Surveys*, vol. 30, no. 2, pp. 170-231, 1998.

[25] J.A.G. Gendrano, B.C. Huang, J.M. Rodrigue, B. Moon, and R.T. Snodgrass, "Parallel Algorithms for Computing Temporal Aggregates," *Proc. Int'l Conf. Data Eng.*, pp. 418-427, Mar. 1999.

[26] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss, "Optimal and Approximate Computation of Summary Statistics for Range Aggregates," *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pp. 227-236, May 2001.

[27] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss, "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries," *Proc. VLDB Conf.*, pp. 79-88, Sept. 2001.

[28] J. Gray, *The Benchmark Handbook for Database and Transaction Processing Systems.* Morgan Kaufmann, 1991.

[29] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub-Totals," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 29-53, 1997.

[30] S. Guha, N. Koudas, and K. Shim, "Data-Streams and Histograms," *Proc. Ann. ACM Symp. Theory of Computing*, pp. 471-475, July 2001.

[31] P.J. Haas and J.M. Hellerstein, "Ripple Joins for Online Aggregation," *Proc. ACM-SIGMOD Conf.*, pp. 287-298, June 1999.

[32] M. Hadjieleftheriou, G. Kollios, and V.J. Tsotras, "Efficient Indexing of Spatiotemporal Objects," *Proc. Conf. Extending Database Technology*, pp. 251-268, Mar. 2002.

[33] J.M. Hellerstein, P.J. Haas, and H.J. Wang, "Online Aggregation," *Proc. ACM-SIGMOD Conf.*, pp. 171-182, May 1997.

[34] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant, "Range Queries in OLAP Data Cubes," *Proc. ACM-SIGMOD Conf.*, pp. 73-88, May 1997.

[35] M. Hogeweg, "Spatio-Temporal Visualization and the Need for Integration," *GeoInformatics*, pp. 32-35, June 2001.

[36] C.S. Jensen and R.T. Snodgrass, "Semantics of Time-Varying Information," *Information Systems*, vol. 21, no. 4, pp. 311-352, June 1996.

[37] C.O. Justice, D.K. Hall, and V.V. Salomonson, "The Moderate Resolution Imaging Spectroradiometer (MODIS): Land Remote Sensing for Global Change Research," *IEEE Trans. Geoscience and Remote Sensing*, vol. 36, pp. 1228-1249, 1998.

[38] V. Khatri, S. Ram, R.T. Snodgrass, and G.M O'Brien, "Supporting User-Defined Granularities and Indeterminacy in a Spatiotemporal Conceptual Model," *Annals Math. and Artificial Intelligence*, vol. 36, nos. 1-2, pp. 195-232, 2002.

[39] J.S. Kim, S.T. Kang, and M.-H. Kim, "On Temporal Aggregate Processing Based on Time Points," *Information Processing Letters*, vol. 71, nos. 5-6, pp. 213-220, Sept. 1999.

[40] N. Kline and R.T. Snodgrass, "Computing Temporal Aggregates," *Proc. Int'l Conf. Data Eng.*, pp. 222-231, Mar. 1995.

[41] R.N. Kline, "Aggregation in Temporal Databases," PhD thesis, Univ. of Arizona, May 1999.

[42] A. Klug, "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions," *J. ACM*, vol. 29, no. 3, pp. 699-717, July 1982.

[43] Landsat, "Landsat Project Website," http://landsat7.usgs.gov/index.php, Oct. 2003.

[44] P.-A. Larson, "Data Reduction by Partial Preaggregation," *Proc. Int'l Conf. Data Eng.*, pp. 706-715, 2002.

[45] I. Lazaridis and S. Mehrotra, "Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure," *Proc. ACM-SIGMOD Conf.*, pp. 401-412, May 2001.

[46] J.A. Cotelo Lema and R.H. Guting, "Dual Grid: A New Approach for Robust Spatial Algebra Implementation," *GeoInformatica*, vol. 6, no. 1, pp. 57-76, 2002.

[47] N. Mamoulis and D. Papadias, "Selectivity Estimation of Complex Spatial Queries," *Proc. Int'l Symp. Advances in Spatial and Temporal Databases*, pp. 155-174, July 2001.

[48] U. Manber, *Introduction to Algorithms: A Creative Approach.* Reading, Mass.: Addison-Wesley, 1989.

[49] Y. Manolopoulos, Y. Theodoridis, and V.J. Tsotras, *Advanced Database Indexing.* Boston: Kluwer Academic, 2000.

[50] J. Melton, *Advanced SQL:1999. Understanding Object-Relational and Other Advanced Features.* San Francisco: The Morgan Kaufman Series in Data Management Systems, Morgan Kaufmann, 2003.

[51] MODIS, MODIS Web, http://modis.gsfc.nasa.gov/, Oct. 2003.

[52] B. Moon, I.F. Vega Lopez, and V. Immanuel, "Scalable Algorithms for Large Temporal Aggregation," *Proc. Int'l Conf. Data Eng.*, pp. 145-156, Mar. 2000.

[53] B. Moon, I.F. Vega Lopez, and V. Immanuel, "Efficient Algorithms for Large-Scale Temporal Aggregation," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 3, pp. 744-751, May/June 2003.

[54] P. Ning, X.S. Wang, and S. Jajodia, "An Algebraic Representation of Calendars," *Annals Math. and Artificial Intelligence*, vol. 36, nos. 1-2, pp. 5-38, 2002.

[55] T. Palpanas, R. Sidle, R. Cochrane, and H. Pirahesh, "Incremental Maintenance for Non-Distributive Aggregate Functions," *Proc. VLDB Conf.*, pp. 802-813, Aug. 2002.

[56] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, "Efficient OLAP Operations in Spatial Data Warehouses," *Proc. Int'l Symp. Advances in Spatial and Temporal Databases*, pp. 443-459, July 2001.

[57] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang, "Indexing Spatio-Temporal Data Warehouses," *Proc. Int'l Conf. Data Eng.*, pp. 166-175, 2002.

[58] C. Parent, S. Spaccapietra, and E. Zimanyi, "Spatio-Temporal Conceptual Models: Data Structures + Space + Time," *Proc. ACM Int'l Symp. Advances in Geographic Information Systems (ACM-GIS)*, pp. 26-33, Nov. 1999.

[59] T.B. Pedersen and N. Tryfona, "Pre-Aggregation in Spatial Data Warehouses," *Proc. Int'l Symp. Advances in Spatial and Temporal Databases*, pp. 460-480, July 2001.

[60] D. Pfoser and N. Tryfona, "Requirements, Definitions and Notations for Spatiotemporal Application Environments," *Proc. ACM Int'l Symp. Advances in Geographic Information Systems (ACM-GIS)*, pp. 124-130, Nov. 1998.

[61] K. Porkaew, I. Lazaridis, and S. Mehrotra, "Querying Mobile Objects in Spatio-Temporal Databases," *Proc. Int'l Symp. Advances in Spatial and Temporal Databases*, pp. 59-78, July 2001.

[62] L. Qiao, D. Agrawal, and A. El Abbadi, "RHist: Adaptive Summarization over Continuous Data Streams," *Proc. ACM Int'l Conf. Information and Knowledge Management (ACM-CIKM)*, pp. 469-476, Nov. 2002.

[63] J.F. Roddick, K. Hornsby, and M. Spiliopoulou, "YABTSSTDMR—Yet Another Bibliography of Temporal, Spatial, and Spatio-Temporal Data Mining Research," *Proc. SIGKDD Temporal Data Mining Workshop*, pp. 167-175, 2001.

[64] V.V. Salomonson, W.L. Barnes, P.W. Maymon, H.E. Montgomery, and H. Ostrow, "MODIS: Advanced Facility Instrument for Studies of the Earth as a System," *IEEE Trans. Geoscience and Remote Sensing*, vol. 27, pp. 145-153, 1989.

[65] S. Saltenis and C.S. Jensen, "Indexing of Moving Objects for Location- Based Services," *Proc. Int'l Conf. Data Eng.*, pp. 463-472, 2002.

[66] B. Salzberg, "Access Methods," *ACM Computing Surveys*, vol. 28, no. 1, pp. 117-120, 1996.

[67] T. Sellis, "Research Issues in Spatio-Temporal Database Systems," *Proc. Int'l Symp. Advances in Spatial Databases*, pp. 3-11, July 1999.

[68] R.T. Snodgrass, *Developing Time-Oriented Database Applications in SQL.* San Francisco: Morgan Kaufmann, 2000.

[69] R.T. Snodgrass and I. Ahn, "A Taxonomy of Time in Databases," *Proc. ACM-SIGMOD Conf.*, pp. 236-246, May 1985.

[70] R.T. Snodgrass, S. Gomez, and L.E. McKenzie Jr., "Aggregates in the Temporal Query Language TQuel," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 5, pp. 826-842, Sept./Oct. 1993.

[71] J. Sun, D. Papadias, Y. Tao, and B. Liu, "Querying about the Past, the Present, and the Future in Spatio-Temporal Databases," *Proc. Int'l Conf. Data Eng.*, pp. 202-213, 2004.

[72] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias, "Spatio-Temporal Aggregation Using Sketches," *Proc. Int'l Conf. Data Eng.*, pp. 214-226, 2004.

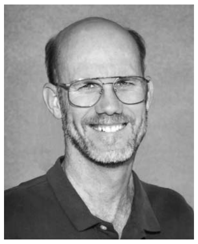
[73] Y. Tao and D. Papadias, "The MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries," *Proc. VLDB Conf.*, pp. 431-440, Sept. 2001.

[74] Y. Tao, D. Papadias, and C. Faloutsos, "Approximate Temporal Aggregation," *Proc. Int'l Conf. Data Eng.*, pp. 190-201, 2004.

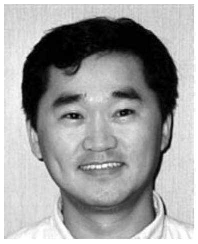
[75] Y. Tao, D. Papadias, and J. Zhang, "Aggregate Processing of Planar Points," *Proc. Conf. Extending Database Technology*, pp. 682-700, Mar. 2002.

[76] Y. Theodoridis, T. Sellis, A.N. Papadopoulos, and Y. Manolopoulos, "Specifications for Efficient Indexing in Spatiotemporal Databases," Technical Report CH-98-01, The Chorochronos research network project, Athens, Greece, Feb. 1998.
[77] Y. Theodoridis, J.R.O. Silva, and M.A. Nascimento, "On the Generation of Spatiotemporal Datasets," *Proc. Int'l Symp. Large Spatial Databases*, pp. 147-164, July 1999.
[78] N. Tryfona and C.S. Jensen, "Conceptual Data Modeling for Spatiotemporal Applications," *GeoInformatica*, vol. 3, no. 3, pp. 245-268, 1999.
[79] P.A. Tuma, "Implementing Historical Aggregates in TempIS," master's thesis, Wayne State Univ., Detroit, Mich., Nov. 1992,
[80] J.W. van Roessel, "Design of a Spatial Data Structure Using the Relational Normal Form," *Int'l J. Geographical Information Systems*, vol. 1, no. 1, pp. 33-50, 1987.
[81] M. Wang, J.S. Vitter, L. Lim, and S. Padmanabhan, "Wavelet-Based Cost Estimation for Spatial Queries," *Proc. Int'l Symp. Advances in Spatial and Temporal Databases*, pp. 175-193, July 2001.
[82] M.F. Worboys, "A Unified Model for Spatial and Temporal Information," *The Computer J.*, vol. 37, no. 1, pp. 26-34, 1994.
[83] W.P. Yan and P.-A. Larson, "Eager Aggregation and Lazy Aggregation," *Proc. VLDB Conf.*, pp. 345-357, Sept. 1995.
[84] J. Yang and J. Widom, "Incremental Computation and Maintenance of Temporal Aggregates," *Proc. Int'l Conf. Data Eng.*, pp. 51-60, Apr. 2001.
[85] X. Ye and J.A. Keane, "Processing Temporal Aggregates in Parallel," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics*, pp. 1373-1378, Oct. 1997.
[86] C. Zaniolo, S. Ceri, C. Faloutsos, R.T. Snodgrass, V.S. Subrahmanian, and R. Zicari, *Advanced Database Systems.* San Francisco: Morgan Kaufmann, 1997.
[87] D. Zhang, "Aggregation Computation over Complex Objects," PhD thesis, Univ. of California, Riverside, Aug. 2002.
[88] D. Zhang, D. Gunopulos, V.J. Tsotras, and B. Seeger, "Temporal Aggregation over Data Streams Using Multiple Granularities," *Proc. Conf. Extending Database Technology*, pp. 646-663, Mar. 2002.
[89] D. Zhang, A. Markowetz, V.J. Tsotras, D. Gunopulos, and B. Seeger, "Efficient Computation of Temporal Aggregates with Range Predicates," *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pp. 237-245, May 2001.
[90] D. Zhang and V.J. Tsotras, "Improving Min/Max Aggregation over Spatial Objects," *Proc. ACM Int'l Symp. Advances in Geographic Information Systems (ACM-GIS)*, pp. 88-93, Nov. 2001.
[91] D. Zhang, V.J. Tsotras, and D. Gunopulos, "Efficient Aggregation over Objects with Extent," *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pp. 121-132, June 2002.

Inés Fernando Vega López received the MS degree in computer science from the University of Arizona in 1999 and the PhD degree in 2004. He holds the BE degree from the Instituto Tecnológico y de Estudios Superiores de Monterrey, Sinaloa México. He is a professor of computer science at the Autonomous University of Sinaloa, México. His current research interests include high-dimensional data indexing and query processing, multimedia, spatial, temporal, and spatiotemporal databases.



Richard T. Snodgrass received the BA degree in physics from Carleton College and the MS and PhD degrees in computer science from Carnegie Mellon University. He joined the University of Arizona in 1989, where he is a professor of computer science. He is an ACM fellow and a senior member of the IEEE and the IEEE Computer Society. He is editor-in-chief of the *ACM Transactions on Database Systems*. He chairs the ACM SIGMOD Advisory Board and has served on the editorial boards of the *International Journal on Very Large Databases* and the *IEEE Transactions on Knowledge and Data Engineering*. He chaired the Americas program committee for the 2001 International Conference on Very Large Databases, chaired the program committees for the 1994 ACM SIGMOD Conference and the 1993 International Workshop on an Infrastructure for Temporal Databases, and was a vice-chair of the program committees for the 1993 and 1994 International Conferences on Data Engineering. He was ACM SIGMOD Chair from 1997 to 2001 and has previously chaired the ACM Publications Board and the ACM SIG Governing Board Portal Committee. He received the ACM SIGMOD Contributions Award in 2002. He chaired the TSQL2 Language Design Committee and edited the book *The TSQL2 Temporal Query Language* (Kluwer Academic Press). He authored *Developing Time-Oriented Database Applications in SQL* (Morgan Kaufmann), was a coauthor of *Advanced Database Systems* (Morgan Kaufmann), and was a coeditor of *Temporal Databases: Theory, Design, and Implementation* (Benjamin/Cummings). He codirects TimeCenter, an international center for the support of temporal database applications on traditional and emerging DBMS technologies. His research interests include temporal databases, query language design, query optimization and evaluation, storage structures, and database design.



Bongki Moon received the MS and BS degrees in computer engineering from Seoul National University, Korea, in 1985 and 1983, respectively, and the PhD degree in computer science from the University of Maryland, College Park, in 1996. He is an associate professor of computer science at the University of Arizona. His current research areas of interest are XML indexing and query processing, high-performance spatial and temporal databases, multidimensional databases, scalable web servers, and parallel and distributed processing. He was a member of the communication systems research staff of Samsung Electronics Corp. and Samsung Advanced Institute of Technology, Korea, from 1985 to 1990. He received the US National Science Foundation CAREER Award in 1999.