Spatiotemporal Aggregate Computation: A Survey

Inés Fernando Vega López, Richard T. Snodgrass, and Bongki Moon

January 14, 2004

TR-77

A TimeCenter Technical Report

Title	Spatiotemporal Aggregate Computation: A Survey
	Copyright © 2004 In´es Fernando Vega L´opez, Richard T. Snodgrass, and Bongki Moon. All rights reserved.
Author(s)	In´es Fernando Vega L´opez, Richard T. Snodgrass, and Bongki Moon
Publication History	January 2004. A TIMECENTER Technical Report.

Contents

1 Introduction **1**

2 Preliminaries **2**
- 2.1 Aggregate Functions 2
- 2.2 Aggregation on Explicit Attributes 3
- 2.3 Aggregation on the Temporal and Spatial Extent 4
- 2.4 Scope of this Survey 4

3 Aggregate Functions on Explicit Attributes **5**
- 3.1 Formal Definition of Aggregation on Explicit Attributes 5
- 3.2 Existing Approaches for Evaluating Aggregate Queries 6
- 3.3 Aggregation and OLAP 7

4 Temporal Aggregates **7**
- 4.1 Formal Definition of Temporal Aggregation 7
- 4.2 Existing Approaches for Evaluating Temporal Aggregate Queries 9
  - 4.2.1 Non-indexed Aggregation Evaluation 10
  - 4.2.2 Indexed Aggregation Evaluation 12
- 4.3 Aggregates on Data Streams 13
- 4.4 Research Opportunities 14

5 Spatial Aggregation **14**
- 5.1 Formal Definition of Spatial Aggregation 15
- 5.2 Existing Approaches for Evaluating Spatial Aggregate Queries 16
- 5.3 Research Opportunities 17

6 Spatiotemporal Aggregation **18**
- 6.1 Formal Definition of Spatiotemporal Aggregation 18
- 6.2 Existing Approaches for Evaluating Spatiotemporal Aggregate Queries 19
- 6.3 Research Opportunities 20

7 Conclusion **21**

8 Acknowledgments **21**

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 of research on types of aggregate queries that have not been studied.
Keywords: Spatiotemporal Databases, Aggregation Queries, Aggregate Function

1 Introduction

A wide variety of scientific and business applications need to capture 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, 69, 80]. We have also observed interest in the organization of data for efficient retrieval [32, 57, 67]. The goal of this paper is to provide a survey on the state of the art techniques for computing spatiotemporal aggregate functions, a topic of fervent interest during the last few years.

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, 85].

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 devoted to spatiotemporal aggregation. Finally, Section 7 summarizes the state of techniques described in this paper and formulates research questions for future work.

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 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, 84]. A pure spatial object can be either a point, a line, or a region in 2 or 3-dimensional space [21, 46, 82]. 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 [80]. Therefore, the combination of the temporal extent with spatial extents of an object leads to different models, ranging from snapshots to 3-dimensional, 4-dimensional, and even 5-dimensional objects depending on the different definitions of time and space [35, 84].

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 in 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. 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].

[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`.

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

Table 1: Employees: A Sample Relation

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

Table 2: The result of an Aggregate query using group composition and the MAX aggregate function

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). The first group is composed the the tuples corresponding to employees Praveen, John, and Frank from whom, Frank has the highest salary (80000). The second group is composed by the tuples corresponding to Fernando and Sofia. Fernando has the highest salary of the group (48000). Therefore, the result for this query is the relation formed by the tuples $\{< 1, 80000 >, < 2, 48000 >\}$.

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 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 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)

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

Table 3: The result of an Aggregate query using partition composition on $DepartmentId$, sliding window composition on Age, and `MAX` aggregate function on $Salary$

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 $Salary$ (45000 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 $\{45000, 42000\}$, resulting in the aggregate value 45000. For the third tuple in the partition, the aggregate function is applied to the set $\{42000, 80000\}$, yielding the aggregate value 80000. 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 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 a 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 spatio-temporal aggregation.

2.4 Scope of this Survey

Aggregation in databases has different meanings depending on the context. For instance, it could refer to aspects of language design, data modeling, or the evaluation of aggregate functions. For data modeling, aggregation is a form of abstraction in which a relationship between objects is considered as a higher level (aggregate) object [63, 64]. The focus of this paper is only on the study of algorithms for computing aggregate functions on spatiotemporal objects. We do not consider issues of language design or data modeling further.

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

Table 4: Evaluation of the SQL-92 Aggregate Functions

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 to use 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. 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 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

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

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 Equation 1. During sliding window composition, for each tuple p in P, a window frame is defined by the following expression [2].

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

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.

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

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 *precede* 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 this methods because they are not critical for understanding spatiotemporal aggregation.

Recently, research work have explored diverse aspects of the aggregation operation. Among them, **optimization**, for applications where performance is of utter importance [44, 85], **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 have 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.

[2] While the SQL standard contemplates the possibility of defi ning 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.

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 pre-computing the aggregated results of different 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 1000 cannot be considered "close" to customer 1001). 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, 88]. 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, 70, 71]. 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 have received several names in the research literature. In particular, we have encountered the terms *span grouping* and *instant grouping*. For span grouping, the time line is partitioned in pre-defined intervals such as year, month, or day [40]. Instant grouping, in the other hand is defined by the data [40, 53, 72]. 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 time line 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 $\mathcal{T}(G, R^T) = \{\tau | \tau \in \mathit{cast}(r[vt], G) \wedge r \in R^T\}$, 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. 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.*

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

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

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

In Equation 2, $r[A_1, \ldots, A_n]$ refers to the explicit attributes of tuple $r \in R^T$. Note that P, as defined by Equation 2, does not generate a strict partition of R^T such as the case of Equation 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, Equation 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*

$$\begin{aligned} \mathit{WF}_{H,SP,J,precedes,follows}(t, R^T) \quad = \quad & \{r | r \in P_{H,SP}(\mathit{cast}(t, H), R^T) \wedge \\ & \mathit{overlaps}(\mathit{cast}(r[vt], J), [t - \mathit{precedes}, t + \mathit{follows}))\}, \end{aligned}$$

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.

$$\mathit{WAgg}_{f_i,H,SP,J,precedes,follows}(R^T) = \{\mathit{cast}(t, H) \circ t \circ f_i(\mathit{WF}_{H,SP,J,precedes,follows}(t, R^T)) | t \in \mathcal{T}(J, R^T)\}$$

For Query 4, we have used the granularity level $year$ to define partitions ($H = year$) and the granularity level $month$ to define window frames ($J = month$). 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. *Non-indexed 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 Non-indexed Aggregation Evaluation

The earliest approach for evaluating temporal aggregation was proposed by Tuma [81]. 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 [72]. 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 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 time stamps 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 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 to use 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 on 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 $value\text{-}length$ 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 time stamp and the other storing the number of tuples ending at the time stamp. By doing an inorder traversal of the tree, the constant intervals can be determined. At the same time, the information on the counters of each node is used to compute the value of the temporal aggregation for each constant 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 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 [87]. They propose to parallelize temporal aggregation queries 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 aggregated 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 non-indexed 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 for temporal aggregation queries was proposed by Yang *et al.* They introduced the *SB-tree* [86] for incrementally computing temporal aggregates using a materialized view approach. This is a disk-based approach that computes temporal aggregates over a base relation that may gradually change by insertion and deletion. The SB-tree contains a hierarchy of intervals associated with partially computed aggregates. 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. 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.

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 *multi-version SB-tree* (MVSB-tree) [91] 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* [91]. 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 pre-aggregation, 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 [76]. This limitation was overcome by Tao *et al.* [76] by 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. Unfortunately, the approaches presented by Zhang *et al.* [91] and Yao etal [76] can only evaluate non-sequenced queries (i.e., the query does not result in a temporal relation). This is because, once a set of tuples have 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.

One drawback of indexed evaluation algorithms is that they either assume that aggregate queries do not include predicates on explicit attributes [86], or they assume that predicates are defined on a single attribute [76, 91]. Furthermore, the approaches presented by Tao *et al.* and Zhang *et al.* [76, 91] ignore the temporal characteristics of the tuples defined by the predicates once they have been selected. For this reason, we do not consider this approach a valid technique for evaluating temporal aggregates. A side effect of ignoring the temporal nature of data is the duplicate count problem, in which temporal objects may be counted more than once.

Another disadvantage of indexed evaluation algorithms is that they can only process certain types of aggregate functions. In particular, partial aggregation, or pre-aggregation, 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 pre-aggregation. 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, 90].

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, 90]. 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.* [90]. 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.* [90] 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.

4.4 Research Opportunities

Temporal aggregation queries can be evaluated by either non-indexed or indexed methods. Non-indexed evaluation methods require a scan of the base relation every time a query is issued. During this process, qualifying tuples are retrieved according to the select predicate and their aggregate information is maintained in a main-memory data structure. At the end of the scan, this data structure will provide a temporal relation with the aggregate values and their corresponding valid intervals. Unfortunately, all evaluation algorithms presented here rely on incrementally maintaining an aggregate value as the base relation is scanned. This approach will not work for non-distributive aggregate functions.

Indexed evaluation methods do not require a scan of the base relation at query time. Instead, before any query is issued, a disk-based data structure that maintains pre-computed aggregate values is created. The evaluation algorithm uses this data structure to answer an aggregate query. Only one of the indexed methods presented here (i.e., [86]) can evaluate a temporal aggregation query resulting in a temporal relation. The rest of the indexed evaluation algorithms evaluate a temporal aggregation query to a non-temporal relation. Because indexed evaluation algorithms rely on pre-aggregation, it is not possible for them to evaluate queries including non-distributive aggregate functions. Furthermore, they provide only limited (or null) support of selection predicates. If a query includes predicates not supported by the disk-based data structure, the query cannot be evaluated.

We have identified the need for indexed evaluation algorithms with extended predicate support. These algorithms should also be capable of evaluating aggregation queries that result in a temporal relation. In addition, both non-indexed and indexed evaluation algorithms should be extended to handle non-distributive aggregate functions.

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 [83]. 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, 68]. Due to the complexity of the spatial operators, the large amount of data, and the difficulty for 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, 83].

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 (i) granules from a spatial granularity do not overlap and (ii) 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 $\mathcal{S}(G, R^S) = \{s | s \in \textit{cast}(r[se], G) \wedge r \in R^S\}$ 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.*

$$\begin{aligned} P_{G,SP}(s, R^S) \quad = \quad & \{t \mid \exists\, r \in R^S \wedge \textit{overlaps}(\textit{cast}(r[se], G), s) \wedge SP(r) \\ & \wedge \;\; t[A_1 \ldots A_n] = r[A_1 \ldots A_n] \wedge \; t[se] = \textit{intersect}(r[se], s)\} \end{aligned} \tag{3}$$

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

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

In Equation 3, $r[A_1 \ldots A_n]$ 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, Equation 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 Equation 3 defines a subset of the rows in R^S rather than a strict partition. Similar to the case of Equation 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

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

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, Equation 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 no 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 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*

$$\begin{aligned} \mathit{WF}_{H,SP,J,windowsize}(s, R^S) \;&=\; \{r | r \in P_{H,SP}(\mathit{cast}(s, H), R^S) \wedge \\ &\qquad \mathit{overlaps}(\mathit{cast}(r[se], J), W_{windowsize}(s))\}, \end{aligned}$$

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

$$\mathit{WAgg}_{f_i,H,J,windowsize}(R^S) = \{\mathit{cast}(s, H) \circ s \circ f_i(\mathit{WF}_{H,J,windowsize}(s, R^S)) | s \in \mathcal{S}(J, R^S)\} \;.$$

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 multi-dimensional 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 multi-dimensional window [45, 77, 83, 93].

In Section 4, we classified existing approaches for the evaluation of temporal aggregates into either non-indexed 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 *et al.* [59] proposed pre-aggregation over spatial data warehouses. They analyze the properties of topological relationships between 2-dimensional spatial objects and show why traditional techniques for pre-aggregation will not work on these settings. Pre-aggregation is a common technique used to efficiently process aggregate functions over data warehouses. However, for pre-aggregation 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 *et al.* presented a methodology to de-compose spatial objects in a way that pre-aggregation can be applied.

Using an approach that combines indexing with pre-aggregation, 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 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, pre-aggregation is possible because they only consider disjoint spatial objects. Zhang *et al.* presented a set of four optimization techniques to improve query performance for `MIN` and `MAX` aggregation [92]. 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.* [93] 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], an 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 multi-dimensional space containing point data items [45]. Their approach uses a tree structure called *Multi-resolution 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.* [77] use a specialized index structure called the *aggregate Point-tree (aP-tree)* for evaluating box aggregation queries over points in 2-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 the number of objects contained by the query window.

5.3 Research Opportunities

All of the approaches presented on this section for the evaluation of spatial aggregation focus on box aggregation queries. In other words, only selection predicates defining a range in space are supported. Any other predicates, such as predicates on explicit attributes, are not supported. In addition, these approaches do not offer support for queries requiring spatial group or partition composition. In fact, once these approaches select the set of qualifying tuples, the spatial properties of the objects represented by these tuples are ignored. Because of this, queries such as Query 5 and Query 6 cannot be evaluated.

We have identified the need for algorithms capable of evaluating spatial aggregation queries that utilize the spatial extent of the objects selected by the query predicates. In addition, such algorithms should be able to provide support for predicates defined on spatial and explicit attributes. Furthermore, support should be provided for non-distributive aggregate functions.

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 applications in the research community. Recent surveys and bibliographic studies show a large amount of research papers on spatiotemporal databases [2, 65]. A spatiotemporal object is an object with both spatial and temporal extent [12, 84]. Not only do the spatial extents of these objects can change over time, but also the values explicit attributes describing non-spatial characteristics of the object may change over time [35, 80].

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, 78, 79]. 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 seasons. Information about spatiotemporal changes in the forest can be obtained by remote sensors such as the *Moderate Resolution Imaging Radio Spectroradiometer (MODIS)* [37, 66] 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 ten 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 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 of 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 $J = km^2 \times (\textit{16-day})$. 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 2 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, 75]. 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 two years. All these methods are based on a disk-based data structure that stores some pre-computed values that are used for answering the queries. Therefore they all are indexed evaluation algorithms.

Zhang *et al.* [91, 89] proposed an extension to their approach for computing aggregates over data streams to handle spatio-temporal 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. Temporal information is stored in a B-tree associated to each region in the R-tree. Index nodes are annotated with aggregate information about the spatial objects contained in the region. This data structure is called *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 3-dimensional R-B-tree* (a3DRB-tree). By keeping pre-aggregate information inside the index, aggregation queries can be answered by intermediate index nodes, thus saving accesses to detailed data.

One drawback of Papadia'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 [74]. Tao *et al.* [74] 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.* [73]. 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 2-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's approach can answer approximate queries to the COUNT aggregate based on a data structure termed *Adaptive Multi-dimensional Histogram (AMH)*, which is updated every time an object transmits a new position. The AMH can only be used for answering snapshots aggregate queries [73].

6.3 Research Opportunities

While the techniques presented in this section are interesting approaches for evaluating spatiotemporal aggregation, they all rely on some form of pre-aggregation. Since they pre-compute results for a particular set of qualifying tuples and keep the summarized information in a data structure, it is not clear whether they could handle aggregate queries that include predicates defined on explicit attributes. Different predicates will select different set of tuples rendering previous pre-computed values useless. Another significant drawback is the fact that none of this approaches can evaluate a spatiotemporal aggregate query resulting in a spatiotemporal relation. This is because they all focus on the box aggregation problem, which after applying spatial and temporal predicates for selecting tuples, ignores the spatial and temporal properties of the qualifying tuples. Problems such as the distinct count problem mentioned by Tao *et al.* [74] would have not arisen if the temporal properties of the selected tuples had been preserved and considered while computing the aggregation.

We have identified the need for algorithms that evaluate a spatiotemporal aggregate queries that perform spatiotemporal group and partition composition. Such queries can identify the time-varying as well as the spatial nature of the aggregation over spatiotemporal relations. That is, the resulting aggregate value is also a spatial object that changes with time. 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. In addition, new approaches should provide support for any kind of selection predicates whether they are defined on the implicit attributes such as spatial and temporal extents or on explicit attributes.

As we have mentioned before, existing approaches for evaluating spatiotemporal aggregate queries rely on pre-aggregation. Pre-aggregation is not useful if the aggregate functions in the query are non-distributive (e.g., MEDIAN), we need to develop algorithms for the efficient evaluation of this type of aggregate functions for spatiotemporal data.

7 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 non-indexed or indexed. Non-indexed 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 pre-aggregation. Hence they only consider distributive aggregate functions such as `COUNT`, `SUM`, and `MAX`. Efficient methods for computing non-distributive 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 [70]. 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 application is not only of interest to know the maximum speed of the wind but also when and where such strong wind is expected.

Lastly, we have also detected that selective predicates are poorly supported by indexed evaluation algorithms. In the best cases, only spatial and temporal predicates defining a box (i.e., box aggregation) are supported. It is important that we develop algorithms for the efficient evaluation of aggregate queries over the full range of spatiotemporal grouping and partition composition.

8 Acknowledgments

This work was sponsored in part by NSF Grant No. IIS-0100436 and NSF Research Infrastructure Program EIA-0080123. It was also supported by Consejo Nacional de Ciencia y Tecnolog´ıa and Universidad Aut´onoma de Sinaloa, scholarship 117476. The authors assume all responsibility for the contents of the paper.

References

[1] Daniel J. Abadi, Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, August 2003.

[2] Tamas Abraham and John F. Roddick. Survey of Spatio-Temporal Databases. *GeoInformatica*, 3(1):61–99, 1999.

[3] Sameet Agarwal, Rakesh Agrawal, Prasad M. Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the Computation of Multidimensional Aggregates. In *Proceedings of the VLDB Conference*, pages 506–521, Bombay, India, September 3–6 1996.

[4] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–16, Madison, WI, June 2002. Invited talk.

[5] Jon L. Bentley. Multidimensional Divide-and-Conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[6] Elisa Bertino et al. *Indexing Techniques for Advanced Database Systems*. Kluwer Academic Publishers, Boston, MA, 1997.

[7] Claudio Bettini, Curtis E. Dyreson, William S. Evans, Richard T. Snodgrass, and Xiaoyang Sean Wang. *Temporal Databases: Research and Practice*, chapter A Glossary of Time Granularity Concepts, pages 406–413. Springer, 1998.

[8] Claudio Bettini, Sushil Jajodia, and Sean X. Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, Berlin, 2000.

[9] Luca Cabibbo and Riccardo Torlone. A Framework for the Investigation of Aggregate Functions in Database Queries. In *Proceedings of the International Conference in Database Theory (ICDT)*, pages 383–397, Jerusalem, Israel, January 1999.

[10] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek Narasayva. Overcoming Limitations of Sampling for Aggregation Queries. In *Proceedings of the International Conference on Data Engineering*, pages 534–542, Heidelberg, Germany, April 2001.

[11] Surajit Chaudhuri, Gautam Das, and Vivek Narasayva. A Robust, Optimization-Based Approach for Approximate Answering of Aggregate Queries. In *Proceedings of the ACM-SIGMOD Conference*, pages 295–306, Santa Barbara, CA, May 2001.

[12] Cindy Xinmin Chen and Carlo Zaniolo. SQL^{ST}: A Spatio-Temporal Data Model and Query Language. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, pages 96–111, Salt Lake City, UT, October 9–12 2000.

[13] Yong-Jin Choi and Chin-Wan Chung. Selectivity Estimation in Spatio-Temporal Queries to Moving Objects. In *Proceedings of the ACM-SIGMOD Conference*, pages 440–451, Madison, WI, June 2002.

[14] Seok-Ju Chun, Chin-Wan Chung, Ju-Hong Lee, and Seok-Lyong Lee. Dynamic Update Cube for Range-Sum Queries. In *Proceedings of the VLDB Conference*, pages 521–530, Roma, Italy, September 11–14 2001.

[15] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate Aggregation Techniques for Sensor Databases. In *Proceedings of the International Conference on Data Engineering*, Boston, USA, March 30–April 2 2004. To appear.

[16] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Streams Statistics over Sliding Windows (Extended Abstract). In *Proceedings of the annual ACM-SIAM Symposium on Discrete Algorithms*, pages 635–644, San Francisco, CA, January 2002.

[17] Alin Dobra, Minos Garofalakis, Johanes Gehrke, and Rajeev Rastogi. Processing Complex Aggregate Queries over Data Streams. In *Proceedings of the ACM-SIGMOD Conference*, pages 61–72, Madison, WI, June 2002.

[18] Curtis E. Dyreson, William S. Evans, Hong Lin, and Richard T. Snodgrass. Efficiently Supporting Temporal Granularities. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):565–587, July–August 2000.

[19] Curtis E. Dyreson, Fabio Grandi, Wolfgang Kafer, Nick Kline, Nikos Lorentzos, Yannis Mitsopoulos, Angelo Montanari, Daniel Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria R. Scalas, Arie Segev, Richard T. Snodgrass, Mike D. Soo, Abdullah Tansel, Paolo Tiberio, Gio Wiederhold, and Christian S. Jensen. A Consensus Glossary of Temporal Database Concepts. *SIGMOD Record*, 23(1):52–64, March 1994.

[20] Robert Epstein. Techniques for Processing of Aggregates in Relational Database Systems. Technical Report UCB/ERL M7918, University of California, Berkeley, CA, February 1979.

[21] Martin Erwing, Ralf H. Guting, Markus Schneider, and Michalis Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3):269–296, 1999.

[22] Luca Forlizzi, Ralf H. Guting, Enrico Nardelli, and Markus Schneider. A Data Model and Data Structures for Moving Object Databases. In *Proceedings of the ACM-SIGMOD Conference*, pages 319–330, Dallas, TX, May 2000.

[23] Johann C. Freytag and Nathan Goodman. Translating Aggregate Queries into Iterative Programs. In *Proceedings of the VLDB Conference*, pages 138–146, Kyoto, Japan, August 1986.

[24] Volker Gaede and Oliver Gunther. Multidimensional Access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[25] Jose Alvin G. Gendrano, Bruce C. Huang, Jim M. Rodrigue, Bongki Moon, and Richard T. Snodgrass. Parallel Algorithms for Computing Temporal Aggregates. In *Proceedings of the International Conference on Data Engineering*, pages 418–427, Sydney, Australia, March 1999.

[26] Anna C. Gilbert, Yannis Lotidis, S Muthukrishnan, and Martin J. Strauss. Optimal and Approximate Computation of Summary Statistics for Range Aggregates. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 227–236, Santa Barbara, CA, May 2001.

[27] Anna C. Gilbert, Yannis Lotidis, S. Muthukrishnan, and Martin J. Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *Proceedings of the VLDB Conference*, pages 79–88, Roma, Italy, September 11–14 2001.

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

[29] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[30] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-Streams and Histograms. In *Proceedings of the annual ACM Symposium on Theory of Computing*, pages 471–475, Hersonissos, Crete, Greece, July 2001.

[31] Peter J. Haas and Joseph M. Hellerstein. Ripple Joins for Online Aggregation. In *Proceedings of the ACM-SIGMOD Conference*, pages 287–298, Philadelphia, PA, USA, June 1–3 1999.

[32] Marios Hadjieleftheriou, George Kollios, and Vassilis J. Tsotras. Efficient Indexing of Spatiotemporal Objects. In *Proceedings of the Conference on Extending Database Technology*, pages 251–268, Prague, Czech Republic, March 25–27 2002.

[33] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online Aggregation. In *Proceedings of the ACM-SIGMOD Conference*, pages 171–182, Tucson, AZ, May 1997.

[34] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range Queries in OLAP Data Cubes. In *Proceedings of the ACM-SIGMOD Conference*, pages 73–88, Tucson, AZ, May 1997.

[35] Marten Hogeweg. Spatio-temporal Visualization and the Need for Integration. *GeoInformatics*, pages 32–35, June 2001.

[36] Christian S. Jensen and Richard T. Snodgrass. Semantics of Time-Varying Information. *Information Systems*, 21(4):311–352, June 1996.

[37] C. Justice, D. Hall, and V. V. Salomonson. The Moderate Resolution Imaging Spectroradiometer (MODIS): Land Remote Sensing for Global Change Research. *IEEE Transactions on Geoscience and Remote Sensing*, 36:1228–1249, 1998.

[38] Vijay Khatri, Sudha Ram, Richard T. Snodgrass, and Grady M O'Brien. Supporting User-defined Granularities and Indeterminacy in a Spatiotemporal Conceptual Model. *Annals of Mathematics and Artificial Intelligence*, 36(1-2):195–232, 2002.

[39] Jong S. Kim, Sung T. Kang, and Myoung-H. Kim. On Temporal Aggregate Processing based on Time Points. *Information Processing Letters*, 71(5-6):213–220, September 1999.

[40] Nick Kline and Richard T. Snodgrass. Computing Temporal Aggregates. In *Proceedings of the International Conference on Data Engineering*, pages 222–231, Taipei, Taiwan, March 1995.

[41] Rodger N. Kline. *Aggregation in Temporal Databases*. PhD thesis, University of Arizona, Tucson, Arizona, May 1999.

[42] Athony Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, 29(3):699–717, July 1982.

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

[44] Per-Ake Larson. Data Reduction by Partial Preaggregation. In *Proceedings of the International Conference on Data Engineering*, pages 706–715, San Jose, CA, February 26–March 1 2002.

[45] Iosif Lazaridis and Sharad Mehrotra. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *Proceedings of the ACM-SIGMOD Conference*, pages 401–412, Santa Barbara, CA, May 2001.

[46] Jose A. Cotelo Lema and Ralf H. Guting. Dual Grid: A New Approach for Robust Spatial Algebra Implementation. *GeoInformatica*, 6(1):57–76, 2002.

[47] Nikos Mamoulis and Dimitris Papadias. Selectivity Estimation of Complex Spatial Queries. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases*, pages 155–174, Redondo Beach, CA, July 12–15 2001.

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

[49] Yannis Manolopoulos, Yannis Theodoridis, and Vassilis J. Tsotras. *Advanced Database Indexing*. Kluwer Academic Publishers, Boston, MA, 2000.

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

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

[52] Bongki Moon, Ines F. Vega Lopez, and Vijaykumar Immanuel. Scalable Algorithms for Large Temporal Aggregation. In *Proceedings of the International Conference on Data Engineering*, pages 145–156, San Diego, CA, March 2000.

[53] Bongki Moon, Ines F. Vega Lopez, and Vijaykumar Immanuel. Efficient Algorithms for Large-Scale Temporal Aggregation. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):744–751, May–June 2003.

[54] Peng Ning, Xiaoyang Sean Wang, and Sushil Jajodia. An Algebraic Representation of Calendars. *Annals of Mathematics and Artificial Intelligence*, 36(1-2):5–38, 2002.

[55] Themistoklis Palpanas, Richard Sidle, Roberta Cochrane, and Hamid Pirahesh. Incremental Maintenance for Non-Distributive Aggregate Functions. In *Proceedings of the VLDB Conference*, pages 802–813, Hong Kong, China, August 2002.

[56] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases*, pages 443–459, Redondo Beach, CA, July 12–15 2001.

[57] Dimitris Papadias, Yufei Tao, Panos Kalnis, and Jun Zhang. Indexing Spatio-Temporal Data Warehouses. In *Proceedings of the International Conference on Data Engineering*, pages 166–175, San Jose, CA, February 26–March 1 2002.

[58] Christine Parent, Stefano Spaccapietra, and Esteban Zimanyi. Spatio-Temporal Conceptual Models: Data Structures + Space + Time. In *Proceedings of the ACM-GIS Conference*, pages 26–33, Kansas City, MO, November 1999.

[59] Torben B. Pedersen and Nectaria Tryfona. Pre-aggregation in Spatial Data Warehouses. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases*, pages 460–480, Redondo Beach, CA, July 12–15 2001.

[60] Dieter Pfoser and Nectaria Tryfona. Requirements, Definitions and Notations for Spatiotemporal Application Environments. In *Proceedings of the ACM-GIS Conference*, pages 124–130, Washington, DC, November 1998.

[61] Kriengkrai Porkaew, Iosif Lazaridis, and Sharad Mehrotra. Querying Mobile Objects in Spatio-Temporal Databases. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases*, pages 59–78, Redondo Beach, CA, July 12–15 2001.

[62] Lin Qiao, Divy Agrawal, and Amr El Abbadi. RHist: Adaptive Summarization over Continuous Data Streams. In *Proceedings of the ACM-CIKM Conference*, pages 469–476, McLean VA, November 2002.

[63] Sudha Ram. Intelligent Database Design Using the Unifying Semantinc Model. *Information and Management*, 29(1995):191–206, 1995.

[64] Sudha Ram and Veda C. Storey. Composite and Grouping: Extending the Realm of Semantic Modeling. In *Proceedings of the Hawaii International Conference on System Sciences HICSS*, pages 212–218, Maui, HA, January 1993.

[65] John F. Roddick, Kathleen Hornsby, and Myra Spiliopoulou. YABTSSTDMR - Yet Another Bibliography of Temporal, Spatial, and Spatio-Temporal Data Mining Research. In *Proceedings of SIGKDD Temporal Data Mining Workshop*, pages 167–175, San Francisco, CA, 2001.

[66] 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 Transactions on Geoscience and Remote Sensing*, 27:145–153, 1989.

[67] Simonas Saltenis and Christian S. Jensen. Indexing of Moving Objects for Location- Based Services. In *Proceedings of the International Conference on Data Engineering*, pages 463–472, San Jose, CA, February 26–March 1 2002.

[68] Betty Salzberg. Access Methods. *ACM Computing Surveys*, 28(1):117–120, 1996.

[69] Timos Sellis. Research Issues in Spatio-Temporal Database Systems. In *Proceedings of the International Symposium on Advances in Spatial Databases*, pages 3–11, Hong Kong, China, July 20–23 1999.

[70] Richard T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, San Francisco, CA, 2000.

[71] Richard T. Snodgrass and Ilsoo Ahn. A Taxonomy of Time in Databases. In *Proceedings of the ACM-SIGMOD Conference*, pages 236–246, Austin, TX, May 1985.

[72] Richard T. Snodgrass, Santiago Gomez, and L. Edwin McKenzie Jr. Aggregates in the Temporal Query Language TQuel. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):826–842, September–October 1993.

[73] Jimeng Sun, Dimitris Papadias, Yufei Tao, and Bin Liu. Querying about the Past, the Present, and the Future in Spatio-Temporal Databases. In *Proceedings of the International Conference on Data Engineering*, Boston, USA, March 30–April 2 2004. To appear.

[74] Yufei Tao, George Kollios, Jeffrey Considine, Feifei Li, and Dimitris Papadias. Spatio-Temporal Aggregation Using Sketches. In *Proceedings of the International Conference on Data Engineering*, Boston, USA, March 30–April 2 2004. To appear.

[75] Yufei Tao and Dimitris Papadias. The MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proceedings of the VLDB Conference*, pages 431–440, Roma, Italy, September 11–14 2001.

[76] Yufei Tao, Dimitris Papadias, and Christos Faloutsos. Approximate Temporal Aggregation. In *Proceedings of the International Conference on Data Engineering*, Boston, USA, March 30–April 2 2004. To appear.

[77] Yufei Tao, Dimitris Papadias, and Jun Zhang. Aggregate Processing of Planar Points. In *Proceedings of the Conference on Extending Database Technology*, pages 682–700, Prague, Czhech Republic, March 25–27 2002.

[78] Yannis Theodoridis, Timos Sellis, Apostolos N. Papadopoulos, and Yannis Manolopoulos. Specifications for Efficient Indexing in Spatiotemporal Databases. Technical Report CH-98-01, The Chorochronos research network project, Athens, Greece, February 1998.

[79] Yannis Theodoridis, Jefferson R. O. Silva, and Mario A. Nascimento. On the Generation of Spatiotemporal Datasets. In *Proceedings of the International Symposium on Large Spatial Databases*, pages 147–164, Hong Kong, China, July 20–23 1999.

[80] Nectaria Tryfona and Christian S. Jensen. Conceptual Data Modeling for Spatiotemporal Applications. *GeoInformatica*, 3(3):245–268, 1999.

[81] Paul A. Tuma. Implementing Historical Aggregates in TempIS. Master's thesis, Wayne State University, Detroit, Michigan, November 1992.

[82] Jan W. van Roessel. Design of a Spatial Data Structure using the Relational Normal Form. *International Journal of Geographical Information Systems*, 1(1):33–50, 1987.

[83] Min Wang, Jeffrey S. Vitter, Lipyeow Lim, and Sriram Padmanabhan. Wavelet-Based Cost Estimation for Spatial Queries. In *Proceedings of the International Symposium on Advances in Spatial and Temporal Databases*, pages 175–193, Redondo Beach, CA, July 12–15 2001.

[84] Michael F. Worboys. A Unified Model for Spatial and Temporal Information. *The Computer Journal*, 37(1):26–34, 1994.

[85] Weipeng P. Yan and Per-Ake Larson. Eager Aggregation and Lazy Aggregation. In *Proceedings of the VLDB Conference*, pages 345–357, Zurich, Switzerland, September 1995.

[86] Jun Yang and Jennifer Widom. Incremental Computation and Maintenance of Temporal Aggregates. In *Proceedings of the International Conference on Data Engineering*, pages 51–60, Heidelberg, Germany, April 2001.

[87] Xinfeng Ye and John A. Keane. Processing Temporal Aggregates in Parallel. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1373–1378, Orlando, FL, October 1997.

[88] Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard T. Snodgrass, V. S. Subrahmanian, and Roberto Zicari. *Advanced Database Systems*. Data Management Systems. Morgan Kaufmann, San Francisco, CA, 1997.

[89] Donghui Zhang. *Aggregation Computation over Complex Objects*. PhD thesis, University of California, Riverside, August 2002.

[90] Donghui Zhang, Dimitris Gunopulos, Vassilis J. Tsotras, and Bernhard Seeger. Temporal Aggregation over Data Streams using Multiple Granularities. In *Proceedings of the Conference on Extending Database Technology*, pages 646–663, Prague, Czhech Republic, March 25–27 2002.

[91] Donghui Zhang, Alexander Markowetz, Vassilis J. Tsotras, Dimitrios Gunopulos, and Bernhard Seeger. Efficient Computation of Temporal Aggregates with Range Predicates. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 237–245, Santa Barbara, CA, May 2001.

[92] Donghui Zhang and Vassilis J. Tsotras. Improving Min/Max Aggregation Over Spatial Objects. In *Proceedings of the ACM-GIS Conference*, pages 88–93, Atlanta, GA, November 2001.

[93] Donghui Zhang, Vassilis J. Tsotras, and Dimitrios Gunopulos. Efficient Aggregation Over Objects with Extent. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 121–132, Madison, WI, June 2002.