Multi-dimensional Aggregation for Temporal Data

Michael Böhlen[1], Johann Gamper[1], and Christian S. Jensen[2]

[1] Free University of Bozen-Bolzano, Italy
{boehlen, gamper}@inf.unibz.it
[2] Aalborg University, Denmark
csj@cs.aau.dk

Abstract. Business Intelligence solutions, encompassing technologies such as multi-dimensional data modeling and aggregate query processing, are being applied increasingly to non-traditional data. This paper extends multi-dimensional aggregation to apply to data with associated interval values that capture when the data hold. In temporal databases, intervals typically capture the states of reality that the data apply to, or capture when the data are, or were, part of the current database state.

This paper proposes a new aggregation operator that addresses several challenges posed by interval data. First, the intervals to be associated with the result tuples may not be known in advance, but depend on the actual data. Such unknown intervals are accommodated by allowing result groups that are specified only partially. Second, the operator contends with the case where an interval associated with data expresses that the data holds for each point in the interval, as well as the case where the data holds only for the entire interval, but must be adjusted to apply to sub-intervals. The paper reports on an implementation of the new operator and on an empirical study that indicates that the operator scales to large data sets and is competitive with respect to other temporal aggregation algorithms.

1 Introduction

Real-world database applications, e.g., in the financial, medical, and scientific domains, manage temporal data, which is data with associated time intervals that capture some temporal aspect of the data, typically when the data were or is true in the modeled reality or when the data was or is part of the current database state. In contrast to this, current database management systems offer precious little built-in query language support for temporal data management.

In step with the increasing diffusion of business intelligence, aggregate computation becomes increasingly important. An aggregate operator transforms an argument relation into a summary result relation. Traditionally this is done by first partitioning the argument relation into groups of tuples with identical values for one or more attributes, then applying an aggregate function, e.g., sum or average, to each group in turn. For interval-valued databases such as temporal databases, aggregation is more complex because the interval values can also be used for defining the grouping of argument tuples.

In this paper we propose a new temporal aggregation operator, the Temporal Multi-Dimensional Aggregation (TMDA) operator. It generalizes a variety of previously

Y. Ioannidis et al. (Eds.): EDBT 2006, LNCS 3896, pp. 257–275, 2006.

proposed aggregation operators and offers orthogonal support for two aspects of aggregation: a) the definition of result groups for which to report one or more aggregate values and b) the definition of aggregation groups, i.e., collections of argument tuples that are associated with the result groups and over which the aggregate functions are computed. Our work builds on recent advances in multi-dimensional query processing [1, 2, 3] and is the first work to leverage these techniques to interval-valued data, in this paper exemplified by temporal data. We provide an efficient implementation of the TMDA operator with an average complexity of $n \log m$, where n is the number of argument tuples and m is the number of result groups. In experimental evaluations on large data sets, the operator exhibits almost linear behavior.

Aggregation of temporal data poses new challenges. Most importantly, the time intervals of result tuples can depend on the actual data and are not known in advance. Therefore, the grouping of the result tuples can only be specified *partially*. Next, aggregation should support what is termed *constant*, *malleable*, and *atomic* semantics of the association between data and time intervals. For example, the association of a time interval with an account balance is constant, meaning that the balance holds for each subinterval of the interval. In contrast, consider the association of a particular week with the number of hours worked by an employee during that week, e.g., 40 hours. Here, the association is malleable, as the 40 hours are considered an aggregate of the hours worked by the employee during each day during that week. An association is atomic if the data cannot be associated with modified timestamps. For example, chemotherapies used in cancer treatment often prescribe a specific amount of a drug to be taken over a specific time period. Neither the amount of the drug nor the time period can be modified without yielding a wrong prescription. All approaches so far support only constant semantics. Finally, a temporal aggregation result might be larger than the argument relation. Specifically, for instantaneous temporal aggregates that are grouped into so-called constant intervals, the result relation size can be twice that of the argument. To quantify the result size, the paper defines the notion of an aggregation factor; and to control the aggregation, the ability to specify fixed time intervals for the result tuples is included.

The rest of the paper is organized as follows. Section 2 studies related work and Sect. 3 covers preliminaries. In Sect. 4, after an analysis of some aggregation queries, we introduce the new TMDA operator. Section 5 presents the implementation of the operator. In Sect. 6, we discuss various properties of this operator including computational complexity and expressiveness. Section 7 reports on an experimental study, and Sect. 8 concludes and offers research directions.

2 Related Work

The notions of instantaneous and cumulative temporal aggregates have been reported previously. The value of an *instantaneous* temporal aggregate at chronon t is computed from the set of tuples that hold at t. The value of a *cumulative* temporal aggregate (also called moving-window aggregate) at chronon t is computed from the set of tuples that hold in the interval $[t-w, t]$, $w \geq 0$. Identical aggregate results with consecutive chronons are coalesced into so-called constant intervals. Most research has been done for instantaneous aggregates, e.g. [4, 5], and cumulative aggregates [6] and temporal aggregates with additional range predicates [7] have received only little attention.

An early proposal by Tuma [8] for computing temporal aggregates requires two scans of the input relation—one for the computation of the time intervals of the result tuples and one for the computation of the aggregates.

A proposal by Kline and Snodgrass [4] scans the input relation only once, building an *aggregation tree* in main memory. Since the tree is not balanced, the worst case time complexity is $O(n^2)$ for n tuples. An improvement, although with the same worst case complexity, is the k-ordered aggregation tree [4]. This approach exploits partial ordering of tuples for garbage collection of old nodes.

Moon et al. [5] use a *balanced tree* for aggregation in main memory that is based on timestamp sorting. This solution works for the functions *sum*, *avg*, and *cnt*; for *min* and *max*, a merge-sort like algorithm is proposed. Both algorithms have a worst case complexity of $\mathcal{O}(n \log n)$. For secondary memory, an efficient bucket algorithm is proposed that assigns the input tuples to buckets according to a partitioning of the time line and also affords long-lived tuples special handling. Aggregation is then performed on each bucket in isolation. The algorithm requires access to the entire database three times.

The *SB-tree* of Yang and Widom [6] supports the disk-based computation and maintenance of instantaneous and cumulative temporal aggregates. An SB-tree contains a hierarchy of intervals associated with partially computed aggregates. With the SB-tree, aggregate queries are applied to an entire base relation—it is not possible to include selection predicates. The *multi-version SB-tree* [7] aims to support aggregate queries coupled with range predicates. A potential problem is that the tree might be larger than the input relation. Tao et al. [9] propose an approximate solution that uses less space than the multi-version SB-tree. Both approaches are restricted to range predicates over a single attribute, and the time interval of an input tuple is deleted once it is selected; hence the result is not a temporal relation.

The existing approaches share three properties. First, the temporal grouping process couples the partitioning of the time line with the grouping of the input tuples. The time line is partitioned into intervals, and an input tuple belongs to a specific partition if its timestamp overlaps that partition. Second, the result tuples are defined for time points and not over time intervals. Third, they allow the use of at most one non-temporal attribute for temporal grouping.

Our TMDA operator, which extends the multi-dimensional join operator [3] to support temporal aggregation, overcomes these limitations and generalizes the aggregation operators discussed above. It decouples the partitioning of the timeline from the grouping of the input tuples, thus allowing to specify result tuples over possibly overlapping intervals and to control the size of the result relation. Furthermore, it supports multiple attribute characteristics. For an efficient implementation we exploit the sorting of the input relation similar to what is done in the k-ordered tree approach [4].

3 Preliminaries

3.1 Notation

We assume a discrete *time domain*, D^T, where the elements are termed chronons (or time points), equipped with a total order $<^T$. Calendar months with the order $<$ satisfy

these requirements, and we use these as our time domain. A timestamp (or time interval) is a convex set over the time domain and is represented by two chronons, $[T_s, T_e]$, denoting its inclusive starting and ending points, respectively. We will use T as a shorthand for $[T_s, T_e]$. For timestamps, we introduce several relations: $t \in T$ means that chronon t is included in timestamp T. For two timestamps T and T', $T' \subseteq T$ iff all chronons in T' are also in T, and $T \cap T'$ returns the set of chronons in both timestamps. If $T \cap T' \neq \emptyset$, we say that the two intervals overlap (or intersect).

A *relation schema* is a three-tuple $S = (\Omega, \Delta, dom)$, where Ω is a non-empty, finite set of attributes, Δ is a finite set of domains, and $dom : \Omega \rightarrow \Delta$ is a function that associates a domain with each attribute. A *temporal relation schema* is a relation schema with at least one timestamp valued attribute (the domain of timestamps belongs to Δ). For the purpose of this paper, we define temporal relation schemas $R = (A_1, \ldots, A_n, T)$ and $G = (B_1, \ldots, B_m, T)$. Note that the assumption that relations have a timestamp attribute T is just for convenience. There is no implicit time attribute, and all definitions are parametrized with a timestamp attribute. As usual, the rename operator ρ can be used to adjust schemas as appropriate.

A *tuple over schema* $S = (\Omega, \Delta, dom)$ is a function $r : \Omega \rightarrow \cup_{\delta \in \Delta} \delta$, such that for every attribute A of Ω, $r(A) \in dom(A)$. A tuple is temporal iff its schema is temporal. To simplify notation we assume an ordering of attributes and represent a tuple as $r = (v_1, \ldots, v_n, t)$. An *relation over schema* R is a finite set of tuples over R, denoted as $\mathbf{r}$. We will also use a couple of shorthands: For a tuple r and an attribute A we write $r.A$ to denote the value of the attribute A in r. For a set of attributes $A_1, \ldots, A_m$, $m < n$, we define $r[A_1, \ldots, A_m] = (r.A_1, \ldots, r.A_m)$.

3.2 Attribute Characteristics

We distinguish among three semantics of the association of a non-timestamp attribute with a timestamp attribute. For a relation with schema $(A_1, \ldots, A_n, T)$ the attribute characteristics wrt. T are given as $C_T = (c_1, \ldots, c_n)$, where $c_i \in \{c, m, a\}$. The values c, m, and a denote constant, malleable, and atomic characteristics, respectively. For example, $C_T = (c, m)$ for the schema (N, H, T) means that N has a constant characteristic and H has a malleable characteristic. If several temporal attributes are used, e.g., valid time and transaction time, a non-timestamp attribute can have different characteristics for the two timestamps. For the rest of the paper, we use only one time attribute T, and C refers to the characteristics wrt. T.

Definition 1. *(*Adjustment of Attribute Values*) Let $r = (v_1, \ldots, v_n, t)$ be a tuple over schema $(A_1, \ldots, A_n, T)$, I be a timestamp, and let $C = (c_1, \ldots, c_n)$ be the attribute characteristics. The adjustment of attribute values is defined as follows:*

$$adj(r, I, C) = (adj(r.A_1, r.T, I, c_1), \ldots, adj(r.A_n, r.T, I, c_n), I)$$

$$adj(v, T, I, c) = \begin{cases} v & \textit{iff } c = \text{'c'} \\ v * |I \cap T|/|T| & \textit{iff } c = \text{'m'} \\ v & \textit{iff } c = \text{'a'} \wedge T = I \\ UNDEF & \textit{iff } c = \text{'a'} \wedge T \neq I \end{cases}$$

Considering the characteristics C, the adj function adjusts each non-timestamp attribute value of r to the time interval I and returns the adjusted tuple. For example, for the tuple $(Jan, 2000, [2003/01, 2003/12])$, the characteristics (c, m), and the time interval $[2003/01, 2003/06]$ the adj function returns $(Jan, 1000, [2003/01, 2003/06])$.

4 The Temporal Multi-dimensional Aggregate Operator

4.1 Temporal Aggregate Examples

As a running example, consider the project database in Fig. 1. The relation EMPL captures project assignments by recording the name of an employee (N), a contract identifier (CID), the department responsible for an assignment (D), the name of a project (P), the hours an employee is assigned to a project (H), a monthly salary (S), and the valid time (T) over which a tuple holds true. The attribute H is malleable, while all other attributes are constant. Figure 2 illustrates relation EMPL together with the intended results for the aggregation queries considered next.

EMPL

	N	CID	D	P	H	S	T
$r1$	Jan	140	DB	P1	2400	1200	[2003/01,2004/03]
$r2$	Jan	163	DB	P1	600	1500	[2004/07,2004/09]
$r3$	Ann	141	DB	P2	500	700	[2003/01,2003/05]
$r4$	Ann	150	DB	P1	1000	800	[2003/06,2004/03]
$r5$	Ann	157	DB	P1	600	500	[2004/01,2004/12]
$r6$	Sue	142	DB	P2	400	800	[2003/01,2003/10]
$r7$	Tom	143	AI	P2	1200	2000	[2003/04,2003/10]
$r8$	Tom	153	AI	P1	900	1800	[2004/01,2004/06]

Fig. 1. Relation EMPL of the Project Database

Query 1: *For each department, compute the total amount of hours spent in projects and the maximal monthly salary.* This instantaneous aggregation groups the result tuples by the non-temporal attribute D. The timestamps of the result tuples are not specified in the query, but are derived from the relation. Hence, the size of the result is data dependent and might exceed that of the input relation, as is the case here.

Query 2: *For each department and year, compute the total hours spent in projects and the maximal monthly salary.* This query differs from the previous one in that the result tuples are grouped according to fixed, user-specified time intervals. This query controls the size of the result relation, which is at the heart of aggregate functions. To the best of our knowledge, this type of query has not been studied previously.

Query 3: *Compute the moving average of hours spent for all six-month periods.* This moving-window query slides in steps of fixed duration over the time line, computing an aggregate for each six-month interval. Unlike for the traditional moving-window operator, the result tuples are valid over time intervals rather than at single time instants.

Query 4: *For the entire lifespan of each department, compute the total hours spent in projects and the maximal monthly salary.* This query specifies a single value for the entire lifespan of a relation.

4.2 Definition of Result Groups and Aggregation Groups

A general temporal aggregation operator should support the specification of two orthogonal aspects of aggregation: definition (1) of the result groups for which to report aggregate results and (2) of the sets of tuples, termed aggregation groups, to associate with each result group and over which to compute the aggregates result(s) to be reported for each group.

Each *result group* can be represented as a tuple in a temporal ("group") relation $\mathbf{g}$ with schema $(B_1, \ldots, B_m, T)$. Each B_i is an attribute from the relation that is the argument of the aggregation operator, and the tuples assume values from B_i that occur in the argument relation. For the timestamp, there are two cases: constant intervals and fixed intervals. With *constant intervals*, the timestamp attribute assumes as values the maximal, non-overlapping intervals over which the set of argument tuples is constant.

Definition 2. (Constant Intervals) *Let* $\mathbf{r}$ *be a temporal relation with timestamp attribute* T*. We define the constant intervals of* $\mathbf{r}$ *as*

$$CI(\mathbf{r}) = \{\, T \mid \forall r \in \mathbf{r}(r.T \supseteq T \vee r.T \cap T = \emptyset) \wedge \\ \forall T' \supset T(\exists r \in \mathbf{r}(r.T \not\supseteq T' \wedge r.T \cap T' \neq \emptyset))\}$$

The first line ensures that result intervals do not cross boundaries of argument intervals. The second ensures that the result intervals are maximal. The constant intervals for the EMPL relation grouped by department are shown in Fig. 2 (Result of Query 1).

Theorem 1. *(*Cardinality of Constant Intervals*) For a temporal relation* $\mathbf{r}$ *with* n *tuples,* $n > 0$*, the cardinality of constant intervals is limited by the following formula:*

$$\mid CI(\mathbf{r}) \mid \leq 2n - 1$$

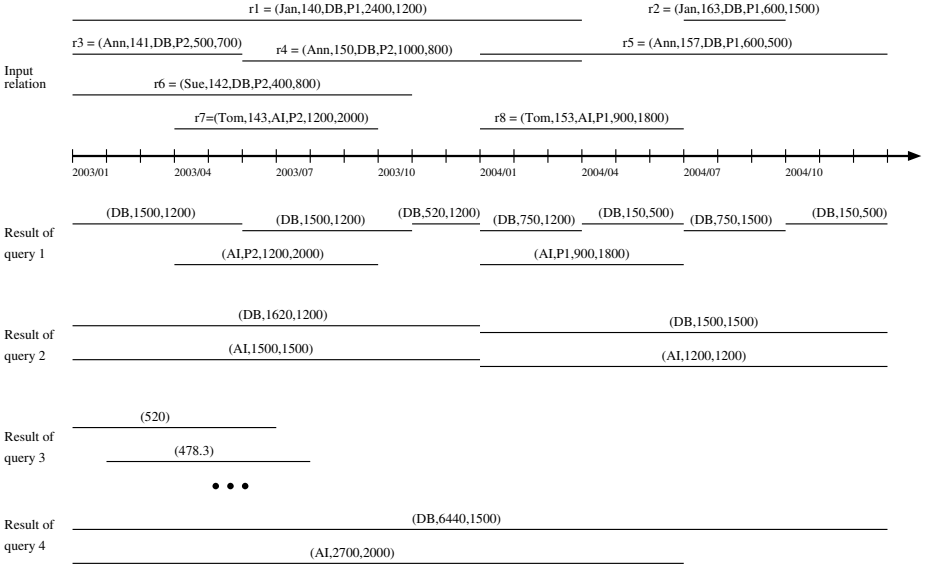


Fig. 2. Temporal Aggregation

Proof. The end points of the tuple's timestamps can be linearly ordered giving at most $2n$ timepoints. n timepoints produce at most $n-1$ consecutive time interals.

The option *fixed intervals* is used when specifying fixed, possibly overlapping, time intervals. Queries 2 and 3 in our running example use different flavors of fixed intervals (see also Fig. 2).

Definition 3. (Fixed Intervals) *Let* $\mathbf{r}$ *be a temporal relation with timestamp attribute* T. *We define the fixed intervals as a user-specified set of timestamps,* $FI(\mathbf{r})$*, that satisfies the following condition:* $\forall T \in FI(\mathbf{r})(\exists r \in \mathbf{r}(r.T \cap T \neq \emptyset))$.

This condition states that fixed intervals must intersect with intervals in the argument relation. The explicit specification of result groups with fixed intervals allows control of the cardinality of the result relation. We use the following definition for quantifying the result size relative to the argument size.

Definition 4. (Aggregation factor) *The aggregation factor of a temporal aggregation operator is defined as the ratio between the cardinality of the result relation* $\mathbf{z}$ *and the cardinality of the argument relation* $\mathbf{r}$*, i.e.,* $af = |\mathbf{z}|/|\mathbf{r}|$.

This factor is between 0 and 1 if the result relation is smaller than the argument relation, 1 if the result relation has the same size as the argument relation, and > 1 if the result relation is larger than the argument relation. For instance, the aggregation factor is $9/8$ for Query 1 and $4/8$ for Query 2.

Having defined result groups, we associate a set of tuples from the argument relation, called *aggregation group*, with each result group. The aggregate(s) for each group is computed over this set. The aggregation groups can be defined by a condition $\theta(g, r)$ that for each input tuple r decides whether it contributes to result group g or not. The condition can involve non-temporal and timestamp attributes. Important classes of conditions are conjunctions of equality conditions for non-temporal attributes and the $overlaps$ relationship for timestamps.

4.3 Definition of the TMDA Operator

The TMDA operator separates the specification of the result groups from the assignment of the input tuples to these result groups, thus providing an expressive framework for temporal aggregation.

Definition 5. (TMDA operator) *Let* $\mathbf{r}$ *and* $\mathbf{g}$ *be relations with timestamp attribute* T, $\mathbf{F} = \{f_{A_{i_1}}, \ldots, f_{A_{i_p}}\}$ *be a set of aggregate functions over attributes in* $\mathbf{r}$, θ *be a condition over attributes in* $\mathbf{g}$ *and* $\mathbf{r}$, *and let* C *be attribute characteristics for* $\mathbf{r}$. *We define the temporal multi-dimensional aggregation operator as*

$$\mathcal{G}^T[\mathbf{F}][\theta][T][C](\mathbf{g},\mathbf{r}) = \{x \mid g \in \mathbf{g} \wedge \mathbf{r}_g = \{\!\{r' \mid r \in \mathbf{r} \wedge \theta(g,r) \wedge r' = adj(r, g.T, C)\}\!\} \wedge x = g \circ (f_{A_{i_1}}(\pi[A_{i_1}](\mathbf{r}_g)), \ldots, f_{A_{i_p}}(\pi[A_{i_p}](\mathbf{r}_g)))\}$$

where π *is a duplicate-preserving projection.*

Relation $\mathbf{g}$ is the group relation that defines the result groups, or (sub-) tuples that will be expanded into result tuples. Relation $\mathbf{r}$ is the (conventional) argument relation. Predicate θ associates an aggregation group, $\mathbf{r}_g \subseteq \mathbf{r}$, with each $g \in \mathbf{g}$. Thereby, the argument tuples are adjusted to the timestamp of the result group, which is also the timestamp of the result tuple. The aggregation functions $f_{A_{i_1}}, \ldots, f_{A_{i_p}}$ are then computed over each aggregation group. The schema of the result relation is the schema of $\mathbf{g}$ augmented with a column for each aggregate value, which for the scope of this paper are labeled $f_{A_{i_1}}, \ldots, f_{A_{i_p}}$.

Example 1. Query 2 can be expressed as follows: $\mathbf{z} = \mathcal{G}^T[\mathbf{F}][\theta][T][C](\mathbf{g}, \text{EMPL}/\mathbf{r})$, where

$$\begin{aligned} \mathbf{g} &: \text{The two leftmost columns in Fig. 3} \\ \mathbf{F} &= \{sum_H, max_S\} \\ \theta &= (\mathbf{g}.D = \mathbf{r}.D) \wedge overlaps(\mathbf{g}.T, \mathbf{r}.T) \\ C &= (c, c, c, c, m, c) \end{aligned}$$

The group relation $\mathbf{g}$ contains a tuple for each combination of department and year. Aggregate functions sum and max on hours and salary are used. The condition θ associates with a result group those argument tuples that have the same department value as the result group and overlap with the group's timestamp. For example, the aggregation group for the DB department in 2003 consists of the tuples $r1$, $r3$, $r4$, and $r6$.

The attribute H is malleable and is adjusted to the timestamp of the group specification before it is passed on to the aggregate functions. Therefore, $r1$, $r3$, $r4$, and $r6$ contribute to the sum of hours of the DB department in 2003 with the values 1920, 500, 700, and 400, respectively. Attribute S is constant, so the adjustment has no effect.

z

D	T	sum_H	max_S
DB	[2003/01,2003/12]	1620	1000
DB	[2004/01,2004/12]	1500	900
AI	[2003/01,2003/12]	1200	2000
AI	[2004/01,2004/12]	900	1800

Fig. 3. Temporal Aggregation with Fixed Interval Semantics

The result relation is shown in Fig. 3. Each result tuple is composed of a result group tuple extended by a value for each aggregate function. To improve readability these two parts are separated by a vertical line.

4.4 Partial Specification of Result Groups

The definition of the TMDA operator requires a completely specified group relation $\mathbf{g}$. For the constant interval semantics, however, the timestamps of the result tuples are calculated from the argument tuples and are not available in advance. To handle this case, we pass on a relational algebra expression that computes the constant intervals, thus reducing constant intervals to fixed intervals.

$$\mathcal{G}^T[\mathbf{F}][\theta \wedge overlaps(\mathbf{g}.T, \mathbf{r}.T)][T][C](CI(\mathbf{g}', \mathbf{r}, \theta)/\mathbf{g}, \mathbf{r})$$

Now the group relation $\mathbf{g}$ is given as an expression $CI(\mathbf{g}', \mathbf{r}, \theta)$ that computes the constant intervals over the argument relation $\mathbf{r}$ based on a group relation $\mathbf{g}'$ that contains the non-temporal groups. This expression basically completes the non-temporal group relation $\mathbf{g}'$ with the constant intervals, i.e., $\mathbf{g} = \{g[B_1, \ldots, B_m] \circ T \mid g \in \mathbf{g}' \wedge T \in CI(\mathbf{g}', \mathbf{r}, \theta)\}$.

While this reduction of constant interval semantics to fixed interval semantics is sound from a semantic point of view, the computation of constant intervals in advance requires operations such as join and union that are computationally costly, as we will illustrate in the experimental section. To improve on the computational efficiency, we introduce partially specified result groups.

Definition 6. (Partially Specified Result Groups) *A result group with schema* $G = (B_1, \ldots, B_m, T)$ *is partially specified iff the value of the timestamp attribute is not specified. We represent a partially specified result tuple as* $g = (v_1, \ldots, v_m, [*, *])$.

With partially specified result groups in place, we push the completion of the result groups with constant intervals into the algorithm for the evaluation of the temporal multidimensional aggregation operator. The constant intervals are computed on the fly while scanning the data relation for the calculation of the aggregates. The partially specified result tuples are replicated to all constant intervals for the corresponding aggregation groups. The *overlaps* relation from condition θ that assigns the relevant data tuples to the constant intervals is applied implicitly by the evaluation algorithm.

Example 2. To express Query 1 we apply the constant interval semantics with a group relation $\mathbf{g}$ that contains the partially specified result groups $\{(DB, [*,*]), (AI, [*,*])\}$. The query is then expressed as $\mathbf{z} = \mathcal{G}^T[\mathbf{F}][\theta][T][C](\mathbf{g}, \text{EMPL}/\mathbf{r})$, where

$$\mathbf{F} = \{sum_H, max_S\}$$
$$\theta = (\mathbf{g}.D = \mathbf{r}.D)$$
$$C = (c, c, c, c, m, c)$$

The condition θ contains only non-temporal constraints. The aggregation group for department DB contains six input tuples that induce seven constant intervals. For department AI, we have two input tuples and two constant intervals. The result relation is shown in Fig. 4. Unlike in previous approaches [10, 6], we do not coalesce consecutive tuples with the same aggregate values, as illustrated by the two first DB tuples; we keep them separate since their lineage is different.

z

D	T	sum_H	max_S
DB	[2003/01,2003/05]	1500	1200
DB	[2003/06,2003/10]	1500	1200
DB	[2003/11,2003/12]	520	1200
DB	[2004/01,2004/03]	750	1200
DB	[2004/04,2004/06]	150	500
DB	[2004/07,2004/09]	750	1500
DB	[2004/10,2004/12]	150	500
AI	[2003/04,2003/09]	1200	2000
AI	[2004/01,2004/06]	900	1800

Fig. 4. Constant Interval Semantics with Partially Specified Result Groups

Converting the result set produced by the TMDA operator to the traditional format of result sets produced by temporal aggregation operators, where consecutive tuples with the same aggregate value are coalesced, can be achieved easily. Thus, the result sets produced by the TMDA operator retains lineage information, and this additional information is easy to eliminate.

5 Implementation of the TMDA Operator

5.1 Idea and Overview

The implementation of the TMDA operator for constant intervals is based on the following observation: if we scan the argument relation, which is ordered by the interval start

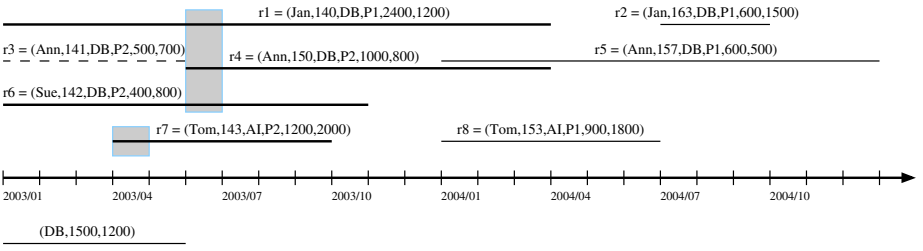


Fig. 5. Processing Input Tuples in TMDA-CI

values of the tuples, we can at any time point t compute the result tuples that end before t (assuming that no tuples that start after t contribute to these result tuples). Hence, as the argument relation is being scanned, result tuples are produced, and old tuples are removed from main memory. Only the tuples that are valid at time point t, termed *open tuples*, are kept in main memory.

Figure 5 illustrates this evaluation strategy for Query 1, showing the situation after reading tuples $r1$, $r3$, $r6$, $r7$, and $r4$, in that order. Thick lines are used for open tuples, dashed lines are used for closed tuples, and solid lines are used for tuples not yet scanned. Grey rectangles indicate the advancement of time. For example, after reading $r4$, the first result tuple for the DB department is computed, $r3$ is closed, and the current time instant for the DB group is $2003/06$; three tuples remain open, and two tuples have not yet been processed. For the AI departement, one tuple is open and one tuple is to be processed.

For the use with fixed intervals, the timestamps of the result tuples are specified in the group relation. So, we do not need to maintain the data tuples in main memory, but can process them and update the aggregate values as we scan the data relation.

In the rest of this section we describe in detail two algorithms for the evaluation of TMDA with constant intervals and fixed intervals, respectively.

5.2 The TMDA-CI Algortihm for Constant Intervals

Figure 6 shows the algorithm, termed TMDA-CI, that evaluates $\mathcal{G}^T$ with constant intervals. The algorithm has five input parameters: the group relation $\mathbf{g}$, the argument relation $\mathbf{r}$, a list of aggregate functions $\mathbf{F} = \{f_{A_{i_1}}, \ldots, f_{A_{i_p}}\}$, a selection predicate θ, and attribute characteristics C. The output is a temporal relation that is composed of $\mathbf{g}$ extended by the values of the aggregate functions in $\mathbf{F}$.

The algorithm uses two types of data structures. A *group table* gt stores each tuple $g \in \mathbf{g}$, together with a pointer to an end-point tree $\mathcal{T}$. An *end-point tree* $\mathcal{T}$ maintains the (potential) end points of the constant intervals together with the relevant attribute values of the currently open tuples. The tree is organized by the end points of the constant intervals, i.e., the end points T_e of the data tuples plus the time points immediately preceding each data tuple. A node with time instant t stores the attribute values $r.A_1, \ldots, r.A_p, r.T_s$ of all data tuples r that end at t. For example, for Query 1 the aggregation tree for the DB department contains a node with time instant $2004/03$ that stores the attribute values of $r1$ and $r4$, i.e.,

Algorithm: TMDA-CI($\mathbf{g}, \mathbf{r}, \mathbf{F}, \theta, C$)

if $\mathbf{g} = \pi[A_1, \ldots, A_m](\mathbf{r})$ **then**
  $gt \leftarrow$ empty group table with columns $B_1, \ldots, B_m, T, \mathcal{T}$;
else
  Initialize gt with $(g, \mathit{empty}\ \mathcal{T}), g \in \mathbf{g}$, and replace timestamp T by $[-\infty, *]$;
Create index for gt on attributes $B_1, \ldots, B_m$; $\mathbf{z} \leftarrow \emptyset$;
foreach *tuple* $r \in \mathbf{r}$ *in chronological order* **do**
  if $\mathbf{g} = \pi[A_1, \ldots, A_m](\mathbf{r})$ *and* $r.A_1, \ldots, r.A_m$ *not yet in* gt **then**
    Insert $(r.A_1, \ldots, r.A_m, [-\infty, *], \mathit{empty}\ \mathcal{T})$ into gt;
  foreach $i \in$ LOOKUP(gt, r, θ) **do**
    if $r.T_s > gt[i].T_s$ **then**
      Insert a new node with time $r.T_s - 1$ into $gt[i].\mathcal{T}$ (if not already there);
      foreach $v \in gt[i].\mathcal{T}$ *in chronological order, where* $v.t < r.T_s$ **do**
        $gt[i].T_e \leftarrow v.t$;
        $\mathbf{z} \leftarrow \mathbf{z} \cup$ RESULTTUPLE$(gt[i], \mathbf{F}, C)$;
        $gt[i].T \leftarrow [v.t + 1, *]$;
        Remove node v from $gt[i].\mathcal{T}$;
    $v \leftarrow$ node in $gt[i].\mathcal{T}$ with time $v.t = r.T_e$ (insert a new node if required);
    $v.\mathit{open} \leftarrow v.\mathit{open} \cup r[A_1, \ldots, A_p, T_s]$;
foreach $gt[i] \in gt$ **do**
  foreach $v \in gt[i].\mathcal{T}$ *in chronological order* **do**
    Create result tuple, add it to $\mathbf{z}$, and close past nodes in $gt[i].\mathcal{T}$;
return $\mathbf{z}$;

Fig. 6. The Algorithm TMDA-CI for Constant Interval Semantics

$(2004/03, \{(2400, 1200, 2003/01), (1800, 800, 2003/06)\})$. A node that stores a potential end point t of a constant interval, but with no tuples ending at t, has an empty data part. For example, tuple $r5$ terminates a constant interval and starts a new one; hence node $(2003/12, \{\})$ will be in the tree. In our implementation we use AVL-trees for end-point trees.

The first step of the algorithm is to initialize the group table gt. If $\mathbf{g}$ is a projection over $\mathbf{r}$, the group table is initially empty and will be populated while scanning the argument tuples. Otherwise, gt is initialized with $\mathbf{g}$, with the start time of the entries set to $-\infty$, and an empty end-point tree is generated for each entry. Finally, an index over the non-temporal attributes is created.

The next step is to process the argument relation $\mathbf{r}$ chronologically with respect to the start times of the tuples. If the group relation is a relational algebra expression, we might have to extend the group table with a new entry before the function LOOKUP determines all result groups to which data tuple r contributes. For each matching result group, two steps are performed: First, if r advances the current time ($r.T_s > gt[i].T_s$), one or more constant intervals can be closed. Chronon $r.T_s - 1$ is a potential end point of a constant interval and is inserted into $gt[i].\mathcal{T}$. Then we process all nodes v in $gt[i].\mathcal{T}$ with $v.t < r.T_s$ in chronological order. Thereby, the timestamp in the group table assumes the constant intervals. We compose the corresponding result tuples and remove the node from the tree. Second, we update the end-point tree with the new data tuple r.

The function LOOKUP gets as input parameters the group table gt, a tuple r, and the selection condition θ. It evaluates the condition θ for all pairs (g, r), $g \in gt$, and returns the indexes of the matching result groups. For the constant interval semantics, an AVL-tree on the non-timestamp attributes is used. For the fixed interval semantics (see algorithm TMDA-FI in Sect. 5.3), we use two AVL-trees, one on the start time and one on the end time of the timestamps. Fixed interval semantics allow us to combine indexes on the timestamps and the non-timestamp attributes.

The algorithm RESULTTUPLE gets as input an entry of the group table $gt[i]$, the set of aggregate functions $\mathbf{F}$, and the attribute characteristics C. It returns the result tuple for the constant interval $gt[i].T$, or the empty set if there are no open tuples in the interval $gt[i].T$. A result tuple is composed of the result group stored in $gt[i]$ extended by the values of the aggregate functions that are computed over all nodes in $gt[i].\mathcal{T}$. The algorithm scans all nodes in the tree, adjusts the attribute values, and computes the aggregate values.

Example 3. We consider the evaluation of Query 1 with algorithm TMDA-CI. Having initialized the group table gt, relation EMPL is processed in chronological order: $r1$, $r3$, $r6$, $r7$, $r4$, $r5$, $r8$, $r2$. For $r1$, function LOOKUP returns the set $\{1\}$. A new node with time $2004/03$ and the attribute values of H, S, and T_s is inserted into $\mathcal{T}_1$, and the start time of the next constant interval is set to $2003/01$ (see Fig. 7a).

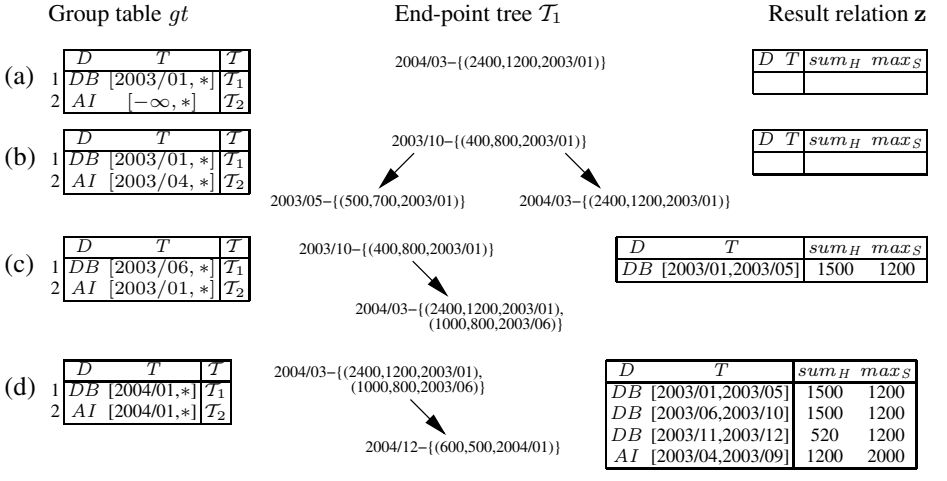


Fig. 7. Evaluation of TMDA-CI after processing $r1, r3, r6, r7, r4, r5$, and $r8$

Figure 7b shows the situation after processing $r1$, $r3$, $r6$, and $r7$. $\mathcal{T}_1$ contains three nodes, $\mathcal{T}_2$ contains one node, and the start time of the next constant interval for the AI result group is set to the start time of $r7$.

The next input tuple is $r4$ for the DB department. Since its start time advances in time, we close the currently open interval and get $[2003/01, 2003/05]$ as the first constant interval for which a result tuple is computed. The adjustment of the attribute values

Algorithm: TMDA-FI($\mathbf{g}, \mathbf{r}, \mathbf{F}, \theta, C$)
if $\mathbf{g} = \pi[A_1, \ldots, A_m, \mathit{cast}(T, G)](\mathbf{r})$ **then**
  $gt \leftarrow$ empty group table with columns $A_1, \ldots, A_m, T, f_{A_{i_1}}, \ldots, f_{A_{i_p}}$;
else
  Initialize gt to $\mathbf{g}$ and extend it with columns $f_{A_{i_1}}, \ldots, f_{A_{i_p}}$ initialized to $NULL$;
Create index for gt on attribute T;
foreach *tuple* $r \in \mathbf{r}$ **do**
  if $\mathbf{g} = \pi[A_1, \ldots, A_m, T](\mathbf{r})$ **then**
    foreach $t \in \mathit{cast}(r.T, G)$ **do**
      Insert $r.A_1, \ldots, r.A_m, t$ into gt if not already there;
  foreach $i \in$ LOOKUP(gt, r, θ) **do**
    $r' \leftarrow$ ADJUST(r, $gt[i].T, C$);
    foreach $f_j \in \mathbf{F}$ **do** $gt[i].f_{A_{i_j}} \leftarrow gt[i].f_{A_{i_j}} \oplus r'.A_{i_j}$;
return gt;

Fig. 8. The Algorithm TMDA-FI for Fixed Interval Semantics

to the constant interval yields $800 + 500 + 200 = 1500$ for the sum function (attribute H is malleable) and $max(1200, 700, 800) = 1200$ for the max function (attribute S is constant). The node with time $2003/05$ is removed from $\mathcal{T}_1$, and the start time of the next constant interval is set to $2003/06$. Finally, the relevant data of $r4$ are added to the already existing node with time $2004/03$. This situation is shown in Fig. 7c.

The next input tuples are $r5$ and $r8$, of which $r5$ contributes to the DB group and gives rise to two result tuples. Tuple $r8$ contributes to the AI group and gives rise to the first result tuple for that group. See Fig. 7d.

5.3 The TMDA-FI Algorithm for Fixed Intervals

Figure 8 shows the TMDA-FI algorithm for the evaluation of operator $\mathcal{G}^T$ with fixed interval semantics. The main data structure is the group table gt that stores the group relation $\mathbf{g}$ and has an additional column labeled $f_{A_{i_j}}$ for each aggregate function $f_{A_{i_j}} \in \mathbf{F}$. The result groups, including their timestamps, are completely specified, so the data tuples need not be stored in an end-point tree, but can be processed as they are read, yielding an incremental computation of the aggregate values.

6 Properties

6.1 Complexity

For the complexity analysis of TMDA-CI, only the processing of the data relation $\mathbf{r}$ is relevant, which is divided into four steps: (possibly) update of the index, lookup in the index, production of result tuples, and insertion of the tuple in the end-point tree.

The update of the index and the lookup in the group table have complexity $\log n_g$, where n_g is the cardinality of the group table. The production of a single result tuple is linear in the number n_o of open tuples. The average number of result tuples induced by

a data tuple depends on the aggregation factor $af = n_z/n_r$, where n_z is the cardinality of the result relation and n_r is the cardinality of the data relation, and on the number of result groups to which r contributes, denoted as $n_{g,r}$. Finally, the insertion of a tuple in an end-point tree has complexity $\log n_o$. This yields an overall time complexity for TMDA-CI of $\mathcal{O}(n_r \max(\log n_g, n_{g,r}\ af\ n_o, \log n_o))$. In general, the size of the data relation n_r might be very large, while all other parameters shall be small. The factor $n_{g,r}$ depends on the selectivity of the condition θ, and is 1 for equality conditions. The aggregation factor, which is between 0 and 2, and the number of open tuples n_o depend mainly on the temporal overlapping of the data tuples. The worst-case complexity is $\mathcal{O}(n_r^2)$ if the start and end points of all data tuples are different and there is a time instant where all tuples hold, hence $n_o = n_r$.

The support for different attribute characteristics comes at a price. For each result tuple, it requires a scan of the entire end-point tree and an adjustment of the attribute values, which becomes a major bottleneck in cases with a large number of open tuples and a high aggregation factor. If only constant attributes were used, the aggregate values could be calculated incrementally similar to [5], as we show later in our experiments.

In the TMDA-FI algorithm, there is no need to maintain the open data tuples, and the aggregate values can be calculated incrementally as the data relation is scanned. The time complexity of TMDA-FI is $\mathcal{O}(n_r\ max(\log n_g, n_{g,r}))$.

6.2 A Spectrum of Temporal Aggregation Operators

The TMDA operator is rather general. The group relation $\mathbf{g}$ is completely independent of the data relation $\mathbf{r}$ and has the only objective to group the results. This arrangement offers enormous flexibility in arranging the results according to various criteria, and it enables the formulation of a range of different forms of temporal aggregates, including the ones proposed previously.

Lemma 1. *(*Aggregation Using Temporal Group Composition [10]*) Let $\mathbf{g}, \mathbf{r}, \mathbf{F}, \theta$, and C be as in Definition 5, SP be a selection predicate over $\mathbf{r}$ as in [10], and let $\mathbf{ch}_G$ be a relation with a single attribute CH that contains all chronons at granularity level G. The operator $\mathcal{G}^T[\mathbf{F}][\theta][T][c, \ldots, c](\mathbf{g}, \mathbf{r})$ with fixed interval semantics simulates aggregation using temporal group composition if $\mathbf{g}$ and θ are defined as follows:*

$$\mathbf{g} = \pi[CH, CH](\mathbf{r} \bowtie [overlaps(T, [CH, CH])]\mathbf{ch}_G)$$
$$\theta = SP(\mathbf{r}) \wedge overlaps(\mathbf{g}.T, \mathbf{r}.T)$$

If the partitioning of the timeline is at the smallest granularity level, temporal group composition simulates instantaneous aggregates [4]; and by transitivity, so does $\mathcal{G}^T$.

Lemma 2. *(*Cumulative Temporal Aggregates [6]*) Let $\mathbf{g}, \mathbf{r}, \mathbf{F}, \theta$, and C be as in Definition 5, let w be a window offset, and let $\mathbf{ch}$ be a relation with a single attribute CH that contains the set of chronons at the lowest granularity level supported by the relation. The operator $\mathcal{G}^T[\mathbf{F}][\theta][T][c, \ldots, c](\mathbf{g}, \mathbf{r})$ with fixed interval semantics simulates cumulative temporal aggregates if $\mathbf{g}$ and θ are defined as follows:*

$$\mathbf{g} = \pi[CH, CH](\mathbf{r} \bowtie [overlaps([T_s, T_e + w], CH)]\mathbf{ch})$$
$$\theta = overlaps([\mathbf{g}.T_e - w, \mathbf{g}.T_e], \mathbf{r}.T)$$

All temporal aggregates developed so far assume a partitioning of the timeline and compute aggregates at time instants. The TMDA operator is more expressive and allows the computation of additional flavors of temporal aggregates. For example, Query 3 is a kind of moving-window aggregate that computes aggregate values over overlapping time intervals. This form of temporal aggregate can easily be expressed by an appropriate group relation.

Another example is the calculation of quarter values that considers data tuples from the corresponding quarter in the past 5 years. In this query, data tuples that contribute to a result tuple are selected from non-contiguous intervals and from outside of the result tuple's timestamp. This functionality has not been addressed in previous temporal aggregation operators. The TMDA operator can afford for such queries by an approriate θ condition.

7 Experimental Evaluation

We carried out a number of experiments with the TMDA-CI and TMDA-FI algorithms, investigating their performance for various settings. All experiments were run on an Intel Pentium workstation with a 3.6 GHz processor and 2 GB memory.

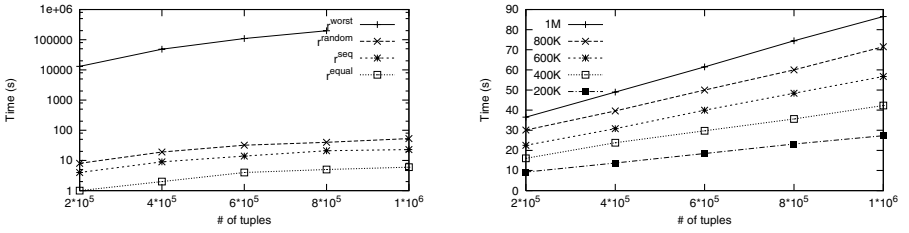
For the experiments, we use data relations that contain from $200,000$ to $1,000,000$ tuples. The lifespan of the data relations is $[0, 2^{25}]$, and we experiment with the following instances [11]:

- $\mathbf{r}^{seq}$: Sequential tuples with one open tuple at each time instant; $af = 1$.
- $\mathbf{r}^{equal}$: All tuples have the same timestamp; $af \in [0.000001, 0.000005]$.
- $\mathbf{r}^{random}$: Start time and duration of the tuples are uniformly distributed in $[0, 2^{25}]$ and $[1, 4000]$, respectively, with 33 open tuples on average; $af \in [1.940553, 1.987989]$.
- $\mathbf{r}^{worst}$: All start and end points are different, and there is a constant interval (in the middle) where all tuples are open; $af \in [1.999995, 1.999999]$.

The group relation contains one entry. This is a worst case since all timestamps end up in the same end-point tree. A group table with more tuples would yield smaller end-point trees and thus better performance.

7.1 Scalability of TMDA-CI and TMDA-FI

The first experiment investigates the scalability of TMDA-CI. Figure 9(a) shows how the time complexity depends on the number of open tuples and the aggregation factor. Relation $\mathbf{r}^{worst}$ with the largest possible number of open tuples and the maximal aggregation factor for constant intervals has a running time that is quadratic in the size of the data relation. For all other data sets, the running time exhibits a linear behavior. Relation $\mathbf{r}^{equal}$ has an aggregation factor close to 0 although the number of open tuples is maximal (however, they are scanned only once). Most of the time (60% for $\mathbf{r}^{random}$ and 97% for $\mathbf{r}^{worst}$) is spent in pruning the end-point tree and computing the result tuples. For each constant interval, the end-point tree must be scanned to adjust malleable attribute values and compute the aggregated value.

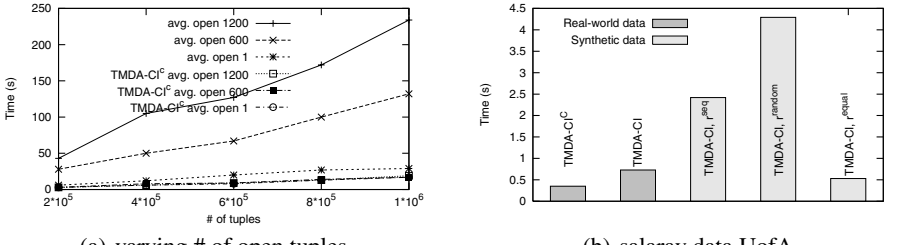


(a) constant intervals (b) fixed intervals

Fig. 9. Evaluation of TMDA-CI and TMDA-FI

The performance of TMDA-FI is not affected by overlapping tuples since the aggregate result is computed incrementally. The key parameter in terms of performance is the number of result groups and the efficiency of the available lookup technique (for each tuple in $\mathbf{r}$, we must find all groups that satisfy the θ condition and therefore have to be updated). Since we use AVL-trees for indexing result groups, the performance increases along with the number of groups, as illustrated in Fig. 9(b). If we used hashing, the lookup time would be constant. However, a hashing only supports equality conditions.

Figure 10 investigates the performance impact of varying the main parameters on the algorithms applied to data relation $\mathbf{r}^{rand}$. Figure 10(a) shows the running time when varying the number of open tuples. The performance decreases since with malleable attributes, all open tuples have to be stored in the end-point tree. As soon as a constant interval has been found, the end-point tree is traversed, the attribute values are adjusted, and the final aggregate is computed. We have also included the performance of a variation of TMDA-CI, denoted TMDA-CI^c, that supports constant attributes only. TMDA-CI^c incrementally computes the aggregates, and its performance is independent of the number of open tuples.



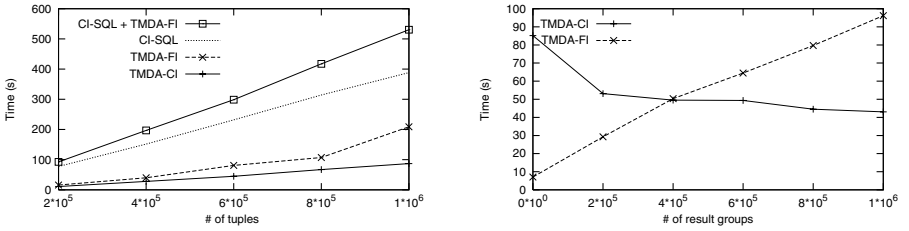
(a) varying # of open tuples (b) salaray data UofA

Fig. 10. Evaluation of TMDA-CI: (a) different data sets, (b) real-world data

Figure 10(b) evaluates the performance for real-world salary data from the University of Arizona. The figure shows that the performance on the real-world data is much better than the performance on most syntetic data sets.

7.2 Constant Versus Fixed Intervals

Figure 11(a) shows the result of computing aggregates with constant interval semantics in two different ways: (1) TMDA-CI with partially specified result groups, and (2) CI-SQL + TMDA-FI, i.e., a priori computation of constant intervals using SQL followed by a call to TMDA-FI. The results confirm that TMDA-CI with partially specified result groups is indeed an efficient way of computing aggregates with constant interval semantics.



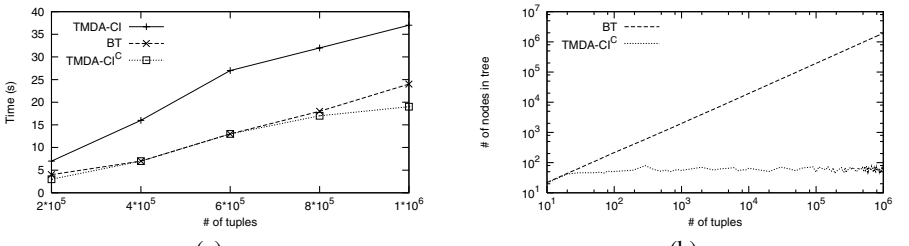
(a) computing constant intervals (b) FI versus CI

Fig. 11. Constant versus Fixed Interval Semantics

Figure 11(b) evaluates TMDA-CI and TMDA-FI for varying result groups. As expected, the performance of TMDA-FI decreases as the number of groups increases. However, up to an aggregation factor of almost 50% FI outperforms CI. Thus, TMDA-FI is efficient for reasonable aggregation factors, and it permits to precisely control the aggregation factor.

7.3 Comparison with the Balanced Tree Algorithm

The last experiment compares TMDA-CI with the balanced-tree algorithm proposed in [5]. This is the most efficient algorithm developed so far, but note that it only handles sum, cnt, and avg—it does not support malleable attributes. Figure 12(a) compares the running time of the balanced-tree algorithm, TMDA-CI, and a modified version, called TMDA-CI^c, that supports only constant attributes and allows incremental computation



(a) (b)

Fig. 12. TMDA-CI versus Balanced Tree

of aggregate values. While TMDA-CI^c has the same performance as the balanced-tree algorithm, the experiments show that the support for multiple attribute characteristics in TMDA-CI is costly and that the DBMS should identify cases when no malleable attributes are present and TMDA-CI^c can be used. The memory consumption of TMDA-CI depends only on the number of open tuples and is much smaller than for the balanced-tree algorithm (see Fig. 12(b)).

8 Conclusion

This paper presents a new aggregation operator, the Temporal Multi-Dimensional Aggregation (TMDA) operator, that leverages recent advances in multi-dimensional query processing [1, 2, 3] to apply to interval-valued data. The TMDA operator generalizes a variety of previously proposed aggregation operators. Most importantly, it clearly separates the definition of result groups from the definition of aggregation groups, i.e., the collections of argument tuples that are associated with the result groups and over which the aggregate functions are computed. This leads to a very expressive framework that allows also to control the size of the result relation. Next, the TMDA operator supports multiple attribute characteristics, including malleable attributes where an attribute value has to be adjusted if the tuple's timestamp changes. Finally, we provid two different algorithms for the evaluation of the TMDA operator with constant intervals and fixed intervals, respectively. Detailed experimental evaluations show that the algorithms are scalable with respect to data set size and compare well with other temporal aggregation algorithms. The evaluation also reveals that the support for multiple attribute characteristics comes at a cost.

Future work includes various optimization steps of the TMDA-CI and TMDA-FI algorithms, including the following ones: optimization rules for relational algebra expressions that interact with the TMDA operator, the initialization of the group table on the fly, indexes on the group table, and a variation of the end-point tree that does not require a totally sorted data relation.

Acknowledgments

We thank Linas Baltrunas and Juozas Gordevicious for the implementation and evaluation of the algortihms. The work was partially funded by the Municipality of Bozen-Bolzano through the eBZ-2015 initiative.

References

1. Akinde, M.O., Böhlen, M.H.: The efficient computation of subqueries in complex OLAP queries. In: Proc. of the 19th Intl. Conf. on Data Engineering, Bangalore, India (2003) 163–174
2. Akinde, M.O., Böhlen, M.H., Johnson, T., Lakshmanan, L.V.S., Srivastava, D.: Efficient OLAP query processing in distributed data warehouses. In: Proc. of the 8th Intl. Conf. on Extending Database Technology, Prague, Czech Republic (2002) 336–353

3. Chatziantoniou, D., Akinde, M.O., Johnson, T., Kim, S.: MD-join: An operator for complex OLAP. In: Proc. of the 17th Intl. Conf. on Data Engineering, Heidelberg, Germany (2001) 524–533
4. Kline, N., Snodgrass, R.T.: Computing temporal aggregates. In: Proc. of the 11th Intl. Conf. on Data Engineering, Taipei, Taiwan (1995) 222–231
5. Moon, B., Vega Lopez, I.F., Immanuel, V.: Efficient algorithms for large-scale temporal aggregation. IEEE Trans. on Knowledge and Data Engineering **15**(3) (2003) 744–759
6. Yang, J., Widom, J.: Incremental computation and maintenance of temporal aggregates. The VLDB Journal **12** (2003) 262–283
7. Zhang, D., Markowetz, A., Tsotras, V., Gunopulos, D., Seeger, B.: Efficient computation of temporal aggregates with range predicates. In: Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Santa Barbara, CA (2001) 237–245
8. Tuma, P.A.: Implementing Historical Aggregates in TempIS. PhD thesis, Wayne State University, Detroit, Michigan (1992)
9. Tao, Y., Papadias, D., Faloutsos, C.: Approximate temporal aggregation. In: Proc. of the 20th Intl. Conf. on Data Engineering, Boston, USA (2004) 190–201
10. Vega Lopez, I.F., Snodgrass, R.T., Moon, B.: Spatiotemporal aggregate computation: A survey. IEEE Trans. on Knowledge and Data Engineering **17**(2) (2005) 271–286
11. Enderle, J., Hampel, M., Seidl, T.: Joining interval data in relational databases. In: Proc. of the ACM SIGMOD Intl. Conf. on Knowledge and Data Engineering, Paris, France (2004) 683–694