The VLDB Journal manuscript No.
(will be inserted by the editor)

Parsimonious Temporal Aggregation

Juozas Gordevičius · **Johann Gamper** · **Michael Böhlen**

Received: date / Accepted: date

Abstract Temporal aggregation is an important operation in temporal databases, and different variants thereof have been proposed. In this paper we introduce a novel temporal aggregation operator, termed parsimonious temporal aggregation (PTA), which overcomes major limitations of existing approaches. PTA takes the result of instant temporal aggregation (ITA) of size n, which might be up to twice as large as the argument relation, and merges similar tuples until a given error (ε) or size (c) bound is reached. The new operator is data-adaptive and allows the user to control the trade-off between the result size and the error introduced by merging. For the precise evaluation of PTA queries, we propose two dynamic programming based algorithms for size- and error-bounded queries, respectively, with a worst-case complexity that is quadratic in n. We present two optimizations that take advantage of temporal gaps and different aggregation groups and achieve a linear runtime in experiments with real-world data. For the quick computation of an approximate PTA answer, we propose an efficient greedy merging strategy with a precision that is upper bounded by $O(\log n)$. We present two algorithms that implement this strategy and begin to merge as ITA tuples are produced. They require $O(n\log(c+\beta))$ time and $O(c+\beta)$ space, where β is the size of a read-ahead buffer and is typically very small. An empirical evaluation on real-world and synthetic data shows that PTA considerably reduces the size of the aggregation result, yet introducing only small errors. The greedy algorithms are scalable for large data sets and introduce less error than other approximation techniques.

Keywords Temporal Aggregation · Data Approximation · Algorithms · Data Mining

Juozas Gordevičius
Vilnius University, Lithuania
Institute of Mathematics and Informatics
E-mail: juozas.gordevicius@mii.vu.lt

Johann Gamper
Free University of Bozen-Bolzano, Italy
Faculty of Computer Science
E-mail: gamper@inf.unibz.it

Michael Böhlen
University of Zurich, Switzerland
Department of Informatics
E-mail: boehlen@ifi.uzh.ch

1 Introduction

1.1 Problem Description

Temporal data are abundant in almost every sector. Whether it is financial, medical, or sensor data, we often associate a timestamp or a validity interval with each record. Temporal aggregation is used to summarize large sets of such data by aggregating specific attribute values over all tuples that hold at a time point or a time interval. As such, temporal aggregation is very important in temporal databases. It has been previously studied in various flavors, most importantly, as instant and span temporal aggregation.

In instant temporal aggregation (ITA) [4,15,18,26,27,30], the aggregate value at a time instant t is computed from the set of all tuples whose timestamp contains t. Result tuples at consecutive time instants with identical aggregate values are then coalesced into result tuples over maximal time intervals during which the aggregate results are constant. While ITA considers the distribution of the data, due to temporally overlapping argument tuples the result size might become up to twice the size of the input [4], which is in conflict with the very idea of aggregation to provide a summary of the data.

Span temporal aggregation (STA) [4,15], on the contrary, allows an application to specify in the query the time intervals for which to report result tuples, e.g., for each year from 2000 to 2005. For each of these intervals a result tuple is produced by aggregating over all argument tuples that overlap with such an interval. Therefore, the result size of STA is predictable, yet it may fail to provide good summaries of the data since the aggregation intervals do not consider the distribution of the input data.

In this paper we introduce parsimonious temporal aggregation (PTA) that overcomes major limitations and combines the best features of instant and span temporal aggregation. By approximating ITA, it computes compact aggregation summaries that reflect the most significant changes in the data over time. If the user specifies the desired result size, PTA minimizes the approximation error. Alternatively, the user can specify a maximal error bound, and PTA minimizes the number of result tuples. Conceptually, PTA operates in two steps: (1) compute the ITA result of the input relation and (2) reduce the ITA result by merging similar and temporally adjacent tuples until a user-specified size or error bound is satisfied. Tuples are adjacent if they belong to the same aggregation group and are not separated by a temporal gap. PTA inherits the data-adaptive approach of ITA and the control over the result size of STA. PTA is useful for such applications as data visualization or similarity search for classification and clustering, where the fine-grained result of ITA is too large to handle and, instead, a concise overview of the data at hand is necessary.

1.2 Example

As a running example throughout the paper we use the relation **proj** in Fig. 1(a), which records information about project assignments: an employee (*Empl*), the project he/she works for (*Proj*), the monthly salary (*Sal*), and the time period (in months) during which the assignment is effective (T). For instance, tuple r_1 states that John works on project A and has a monthly salary of 800 in the time period $[1,4]$. In the graphical illustration timestamps are drawn as horizontal lines.

Consider the following STA query: *"For each project, what is the average monthly salary in each trimester?"*. This query explicitly partitions the time line into trimesters (independent of the distribution of the data) over which the results are reported for each project. The result is shown in Fig. 1(b).

Fig. 1(c) shows the result of the corresponding ITA query: *"For each project, what is the average monthly salary?"*. Here the average salary is determined for each month and project, followed by a coalescing of value-equivalent tuples over consecutive time points. The result size exceeds the size of the input relation, though some adjacent tuples have quite similar aggregate values, e.g., s_4 and s_5.

Fig. 1(d) shows the result of the following PTA query: *"For each project, what is the average monthly salary, where the result size shall not exceed 4 tuples?"*. The result is obtained by applying three merging steps on the ITA result in Fig. 1(c) such that the introduced error is minimal. For instance, s_1 and s_2 are merged into the PTA result tuple z_1, where the average salary of z_1 is computed by averaging the salaries of s_1 and s_2 over each month, i.e., 800 for two months and 600 for one month, yielding the value 733.33. Tuples separated by temporal gaps (e.g., s_6 and s_7) or belonging to different aggregation groups (e.g., s_3 and s_6) cannot be merged. Different from the other two operators, PTA reveals significant changes in the aggregation values.

1.3 Contributions

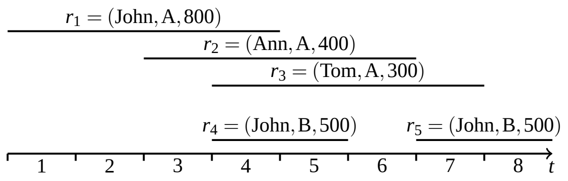
We introduce and formally define a novel temporal aggregation operator, termed parsimonious temporal aggregation (PTA), that overcomes major limitations and combines the best features of previous temporal aggregation operators. Two variants of PTA are provided. Size-bounded PTA allows the specification of a maximal result size, c, while error-bounded PTA allows the specification of a maximal error threshold, ε.

Second, we propose two dynamic programming based algorithms, PTA_c and PTA_ε, for the precise evaluation of size-bounded and error-bounded PTA queries, respectively. The two algorithms require $O(n^2cp)$ time and $O(n^2)$ space in the worst case, where n is the ITA result size, c is the PTA result size, and p is the number of aggregate functions. We present two optimizations that take advantage of temporal gaps and aggregation groups, yielding almost linear running times on real-world datasets.

Third, for a quick evaluation over potentially huge data sets, we propose an efficient greedy merging strategy that computes an approximation of the PTA result with a precision that is upper-bounded by $O(\log n)$. We present two algorithms, $gPTA_c$ and $gPTA_\varepsilon$, for size- and error-bounded queries, respectively, that implement this strategy and tightly integrate the computation of ITA and the merging step. That is, the merging process needs not to wait for the completion of the ITA result, but starts with the merging as ITA result tuples are produced. The algorithms run in $O(n\log(c+\beta))$ time and $O(c+\beta)$ space, where $c+\beta \leq n$; we show experimentally that β is typically very small.

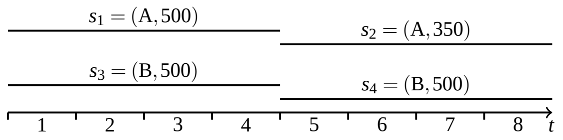
Finally, we conduct extensive experiments using real-world and synthetic data. The results show that PTA achieves a significant reduction of the ITA result, introducing only a small error. The greedy algorithms are scalable for large data sets, and they introduce significantly less error than other state of the art time series and temporal approximation methods.

	Empl	*Proj*	*Sal*	*T*
r_1	John	A	800	[1, 4]
r_2	Ann	A	400	[3, 6]
r_3	Tom	A	300	[4, 7]
r_4	John	B	500	[4, 5]
r_5	John	B	500	[7, 8]


$r_1 = (\text{John}, \text{A}, 800)$
$r_2 = (\text{Ann}, \text{A}, 400)$
$r_3 = (\text{Tom}, \text{A}, 300)$
$r_4 = (\text{John}, \text{B}, 500)$
$r_5 = (\text{John}, \text{B}, 500)$
1 2 3 4 5 6 7 8 t

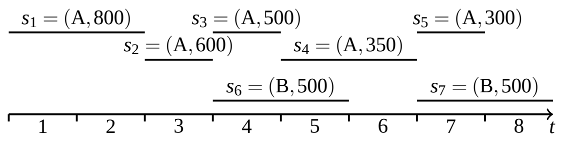
(a) **proj** Relation

	Proj	*AvgSal*	*T*
s_1	A	500	[1, 4]
s_2	A	350	[5, 8]
s_3	B	500	[1, 4]
s_4	B	500	[5, 8]



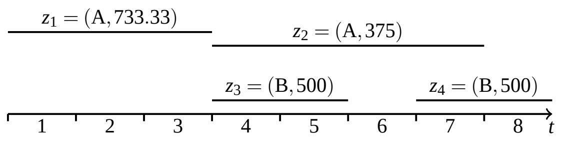
(b) STA Result

	Proj	*AvgSal*	*T*
s_1	A	800	[1, 2]
s_2	A	600	[3, 3]
s_3	A	500	[4, 4]
s_4	A	350	[5, 6]
s_5	A	300	[7, 7]
s_6	B	500	[4, 5]
s_7	B	500	[7, 8]



(c) ITA Result

	Proj	*AvgSal*	*T*
z_1	A	733.33	[1, 3]
z_2	A	375	[4, 7]
z_3	B	500	[4, 5]
z_4	B	500	[7, 8]



(d) PTA Result of Size 4

Fig. 1 The **proj** relation and different temporal aggregation queries to compute the average monthly salary per project.

1.4 Organization

The rest of the paper is organized as follows. In Section 2 we discuss related work. After presenting some preliminary concepts in Section 3, we introduce and formally define the PTA operator in Section 4. In Section 5 we describe the dynamic programming based evaluation algorithms, followed by the greedy evaluation strategy and algorithms in Section 6. Section 7 reports the experimental results. Finally, Section 8 draws conclusions and outlines future work.

2 Related Work

2.1 Temporal Aggregation

Various forms of temporal aggregation have been studied in the past. They differ mainly in how the time line is partitioned. Instant temporal aggregation (ITA) [15,18,27] operates at the smallest granularity of time instants. For each time instant, t, the aggregate functions are computed over all tuples that hold at t. Identical aggregate results at consecutive time instants are coalesced into tuples over maximal time intervals. Moving-window (or cumulative) temporal aggregation (MWTA) [19,23,30] extends ITA and computes for each time instant, t, the aggregate values over all tuples that hold in a window "around" t. While ITA and MWTA report the most detailed result, the main drawback is that the result size might become up to twice as large as the input size.

Span temporal aggregation (STA) [23] allows users to control the result size by partitioning the time line into intervals that are specified in the query. For each such interval, a result tuple is computed over all argument tuples that overlap with that interval. STA does not consider the distribution of the data, and most approaches consider only regular time spans expressed in terms of granularities, e.g., years or months.

Vega Lopez et al. [27] formalize temporal aggregation in a uniform framework that enables the comparison of the different temporal aggregation variants. Similarly, the multi-dimensional temporal aggregation operator [4] generalizes previous temporal aggregation operators towards more flexibility for the specification of aggregation groups.

The approximation of temporal aggregation is a relatively new topic [9,10,25]. Tao et al. [25] were the first to introduce an approximate temporal aggregation technique, which leverages span temporal aggregation and, for a given time interval, finds an approximate aggregation result from tuples that overlap with that interval. The approach uses off-the-shelf B- and R-trees to compute the aggregation result in linear space and logarithmic time with respect to the size of the database. Since the proposed technique approximates span temporal aggregation, where the user specifies the aggregation intervals, it is not data-adaptive and cannot be used to reveal significant changes in the data. Moreover, only error-bounded approximation for the *sum* and *count* aggregation functions is possible.

In our previous work [9,10] we introduce parsimonious temporal aggregation as an approximation of ITA. In this paper we extend it in various directions. First, in addition to size-bounded PTA that reduces an ITA relation to a user-specified size, we define error-bounded PTA that minimizes the result size under a maximal error threshold specified by the user. We provide new query evaluation algorithms for error-bounded PTA, and we report the results of additional experiments, including the comparison of PTA with various time series approximation techniques.

Berberich et al. [2] introduce an approximate temporal coalescing (ATC) technique to reduce the size of a temporal inverted file index. The index is represented as a temporal relation, where each record contains a document reference, a term, its index value, and a validity interval. ATC reads sorted and temporally adjacent tuples that share the same document/term pair and merges them if the introduced local error does not exceed a user-specified threshold. Though the aim is different, ATC can be used to merge ITA result tuples. Experiments show that the total error of ATC is up to an order of magnitude higher than that of PTA and varies significantly depending on the dataset. Such a behavior is not surprising since ATC makes merging decisions based on local information only. The performance gain of ATC with respect to our greedy algorithms is negligible.

2.2 Time Series Approximation

An ITA result can be considered as a time series if no temporal gaps and aggregation groups are present, and hence time series approximation techniques can be applied to obtain an PTA approximation. The need to visualize, mine, and index abundant amounts of time series data has motivated extensive research on their approximate representation [1,6,7,14,16,17,20,22,31]. Lin et al. [17] provide an excellent classification of different representation techniques. In Fig. 2(a) we plot an ITA result over a small excerpt of the Incumbents data set that we use for the experimental evaluation (cf. Sec. 7). With only one aggregate value and no aggregation groups and temporal gaps it can be considered as a time series. Figures 2(b-h) depict different approximate representations of this ITA result. Dotted lines show the ITA result, while solid lines represent the approximated values.

Discrete wavelet transform (DWT) [1] with Haar wavelets recursively averages neighbouring values, yielding a representation of the data at various levels of resolution. Coefficients that allow to reconstruct finer representations from coarser resolutions are stored at each level. A step function constructed from the c most influential coefficients approximates the original time series. DWT might break apart constant-valued segments, which leads to higher approximation errors. Moreover, since the size of the input data has to be a power of two, the data often need to be padded, which influences the approximation result. In Fig. 2(b) an example of a DWT approximation using 10 wavelet coefficients is shown using solid lines. Observe the fluctuation of values at the right-hand side because of padding.

Contrary to wavelets, discrete Fourier transform (DFT) [16] approximates the input time series with a continuous function. An example of a DFT approximation using 10 coefficients is shown in Fig. 2(c). DFT cannot be directly employed to evaluate PTA queries as we require the output of PTA to be a step function with a user-specified number of segments and constant aggregation values throughout each segment. Keogh and Kasetty [12] have shown that the difference in terms of approximation error between DWT and DFT is typically small. In this work we show that the approximation quality of $gPTA_c$ algorithm is much better than that of DWT.

Cai and Ng [6] suggest to use Chebyshev polynomials to represent and index time series data for similarity search. Similarly to DFT, the signal restored from Chebyshev coefficients, see example in Fig. 2(d), is a continuous function. However, instead of total approximation error it minimizes the maximum deviation from the true value. The authors argue that this property is desired in similarity search and define a distance for Chebyshev coefficients that lowerbounds the Euclidean distance for the original time series. They show that a small number of coefficients, up to 25, is enough to construct an index that allows very efficient search. Computing more coefficients may not be practical as the computation time depends linearly on their number. In this work our aim is to minimize the total error of approximation instead of maximum deviation, nevertheless, we compare the time series restored from Chebyshev coefficients to corresponding $gPTA_c$ approximations having the same number of intervals. We show that $gPTA_c$ provides a significantly better approximation.

Piecewise aggregate approximation (PAA) for time series has been introduced by Keogh and Pazzani [14] and, concurrently, by Yi and Faloutsos [31] where it is termed Segmented means. A time series is divided into c segments of equal length, and for each segment the average value is computed. It is shown that PAA and DWT produce the same result under the L_2 norm when c and the length of the input are powers of 2. Like DWT, this approach is not data-adaptive. An example of a PAA approximation using 10 intervals is shown in Fig. 2(e).

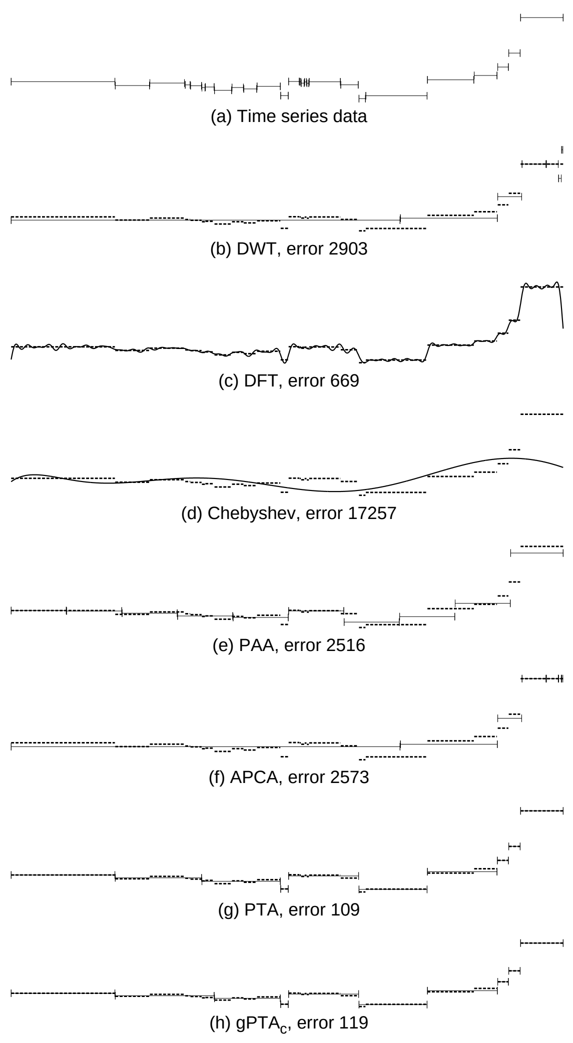


Fig. 2 Various approximations of time series data.

The symbolic aggregate approximation SAX [17] and its scalable version iSAX [22] allow very efficient nearest neighbor queries by representing the input time series as a word of symbols. The symbolic representation is constructed in two steps. First, PAA is used to partition the time series into c equal sized segments. Second, the segments are represented using w different symbols in such a way that each symbol has approximately the same probability of occurrence. Increasing the vocabulary w leads to more precise representations. The limitations of PAA carry over to SAX.

Elmeleegy et al. [8] propose an approach similar to ATC that aims at guaranteeing a given error bound while maximizing the compression ratio of a continuous time series data stream. The proposed algorithm constructs a new approximation segment if attaching an incoming data point to the previous segment exceeds an error threshold. The data in each segment are approximated with a linear function. In line with other stream approximation techniques, the infinity norm is used as error measure. Consequently, the global approximation error is maintained under a given threshold by keeping local errors under the same threshold. However, the infinity norm is not appropriate for PTA, and we use the Euclidean norm as suggested by Jagadish et al. [11]. Additionally, our approach allows the user to specify either a global size or a global error bound.

Chakrabarti et al. [7] leverage PAA and DWT and introduce adaptive piecewise constant approximation (APCA) for time series. The authors suggest to infer approximation segments from the underlying data to increase the approximation accuracy. The proposed algorithm starts by decomposing the input into DWT coefficients. Only the c most significant coefficients are used to reconstruct the time series. Since the reconstruction step may yield up to $3c$ segments, the algorithm greedily merges the most similar ones to reduce the time series to c segments. As illustrated in Fig. 2(f), APCA improves over DWT by inserting true average values into the segments that are inferred from wavelet coefficients. Being data-adaptive it also improves over PAA. APCA still introduces higher errors than our greedy evaluation algorithms. This is due to the underlying DWT decomposition, which is not data-adaptive and breaks apart the constant-value segments in the ITA relation, yielding large approximation errors. The consequent greedy merging step of APCA can only smooth these errors out, but cannot fix them entirely.

Palpanas et al. [20, 21] propose a framework that approximates older time series entries with a higher error and keeps recent entries more precise. The user can specify a relative or absolute amnesic function that controls the amount of error permitted at each time point. With an absolute amnesic function the number of output segments is minimized, with an error at each segment that is upper-bounded by a user-specified threshold. For an absolute amnesic function $AA(t) = \varepsilon$ the amnesic effect is eliminated and the problem becomes equivalent to ATC. Choosing a relative amnesic function transforms the problem into an instance of APCA, where the error has to be minimized subject to the maximal number of allowed segments. The problem is equivalent to size-bounded PTA when a relative amnesic function is used with $RA(t) = 1$, that is, its effect is disabled by permitting an equal amount of amnesia at every time point. Similar to $gPTA_c$ the proposed evaluation algorithm follows a dynamic programming strategy. The two approaches are identical when dealing with one-dimensional time series data and the above amnesic function. In this paper we extend the strategy to multi-dimensional data and take advantage of temporal gaps and aggregation groups to significantly speed up the evaluation. In addition, a greedy evaluation algorithm is presented that merges adjacent tuples whenever more than c entries are present in the heap. $gPTA_c$ follows the same idea. Beyond this we explore various early merging strategies and show that shortly delaying the merging step can significantly reduce the approximation error. For time series data and parameter $\delta = 0$ for $gPTA_c$, the two algorithms are equivalent.

In summary, time series approximation algorithms do not consider the constant value intervals in the ITA result. When applied in the context of temporal aggregation, they might split such intervals and produce high approximation errors. Moreover, temporal gaps in the data and the approximation of multiple aggregation groups under one global (error or size) bound are not supported. PTA employs a more general data model and overcomes such limitations. Its data-adaptive approach outperforms the time series algorithms in terms of approximation quality (cf. Fig. 2 and Sec. 7).

2.3 Histogram Construction

Jagadish et al. [11] present an optimal, dynamic programming based algorithm for the construction of histograms for one-dimensional data, given either a size or error bound. The authors advocate the use of the sum square error and show how to compute it in constant time, which leads to an overall complexity of $O(n^2c)$ time and $O(n^2)$ space for the histogram construction. Our algorithm for the precise computation of PTA emanates from this work and extends it for multi-dimensional data. Furthermore, we exploit the presence of temporal gaps and aggregation groups in our data and propose further optimizations of the dynamic programming scheme.

3 Preliminaries

A *relation schema* is a triple $R = (\Omega, \Delta, dom)$, where Ω is a non-empty, finite set of attributes, Δ is a finite set of domains, and $dom : \Omega \to \Delta$ is a function that associates a domain with each attribute. A *tuple*, r, over schema R is a finite set that contains for every $A_i \in \Omega$ a pair A_i/v_i such that $v_i \in dom(A_i)$. A *relation*, $\mathbf{r}$, over schema R is a finite set of tuples over R. A *temporal relation schema* is a relation schema with at least one timestamp attribute, T, that ranges over the *time domain* Δ^T, i.e., $T \in \Omega$ and $dom(T) = \Delta^T \in \Delta$. For simplicity we assume an ordering of the attributes and represent a temporal relation schema as $R = (A_1, \ldots, A_m, T)$ and a corresponding tuple as $r = (v_1, \ldots, v_m, \mathbf{t})$. 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 $\mathbf{A} = \{A_1, \ldots, A_k\}$, $k \leq m$, we define $r.\mathbf{A} = (r.A_1, \ldots, r.A_k)$.

We assume a discrete time domain, Δ^T. Its elements are termed chronons (or time points/instants), equipped with a total order, $<^T$ (e.g., calendar months with the usual chronological order). A *timestamp* (or time interval), $\mathbf{t}$, is a convex set of chronons over the time domain, and it is represented by two chronons, $[t_b, t_e]$, denoting its inclusive starting and ending points, respectively. If $\mathbf{t} \cap \mathbf{t}' \neq \emptyset$, we say that the two intervals overlap (or intersect), otherwise they are disjoint.

Next, we define instant temporal aggregation.

Definition 1 (Instant Temporal Aggregation) Let $\mathbf{r}$ be a temporal relation with schema $R = (A_1, \ldots, A_m, T)$, grouping attributes $\mathbf{A} = \{A_1, \ldots, A_k\}$, and aggregate functions $\mathbf{F} = \{f_1/B_1, \ldots, f_p/B_p\}$. Furthermore, let *coalesce* be the coalescing operator [5] and $\mathbf{r}_{g,t} = \{r \mid r \in \mathbf{r} \wedge r.\mathbf{A} = g \wedge t \in r.T\}$ be all tuples of $\mathbf{r}$ with grouping attribute values equal to g and intersecting time point t. *Instant temporal aggregation* is defined as

$$\begin{aligned}\mathscr{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r} = coalesce\{s \mid\, & g \in \pi[\mathbf{A}]\mathbf{r} \wedge t \in \Delta^T \wedge \mathbf{r}_{g,t} \neq \emptyset \wedge \\ & f = (f_1(\mathbf{r}_{g,t}), \ldots, f_p(\mathbf{r}_{g,t})) \wedge \\ & s = g \circ f \circ [t,t]\}.\end{aligned}$$

and has schema $S = (A_1, \ldots, A_k, B_1, \ldots, B_p, T)$.

g ranges over all combinations of grouping attribute values in $\mathbf{r}$, and t over the time domain. For each combination of g and t, the aggregation group $\mathbf{r}_{g,t}$ collects all argument tuples that have grouping attribute values equal to g and are valid at time t. A result tuple, s, is produced by extending g with the result of the aggregate functions f_i evaluated over the non-empty $\mathbf{r}_{g,t}$ and with timestamp $[t,t]$. Each f_i is some aggregation function that takes a (temporal) relation as argument and applies aggregation to one of the relation's attributes. The resulting value is stored as the value of an attribute named B_i. The final step is coalescing of value-equivalent result tuples over consecutive time points into tuples over maximal time periods during which the aggregate values do not change. The result of ITA contains up to $2n-1$ tuples, where n is the size of the argument relation [4].

Example 1 The ITA query *"What is the average monthly salary for each project?"* in our running example (see also Fig. 1(c)) is formulated as $\mathscr{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{proj}$ with aggregate functions $\mathbf{F} = \{avg(Sal)/AvgSal\}$ and grouping attributes $\mathbf{A} = \{Proj\}$. The schema of the result relation is $(Proj, AvgSal, T)$.

A property common to any ITA result relation, $\mathbf{s}$, is that the timestamps of the tuples within a single aggregation group do not intersect, i.e., for any pair of tuples $s_i, s_j \in \mathbf{s}$ such that $s_i \neq s_j$ and $s_i.\mathbf{A} = s_j.\mathbf{A}$ we have $s_i.T \cap s_j.T = \emptyset$. We term such (temporal) relations *sequential*. For example, in Fig. 1(c) the timestamps of all tuples with identical *Proj* values are temporally disjoint.

4 Parsimonious Temporal Aggregation

In this section we introduce and define parsimonious temporal aggregation, PTA, which conceptually comprises two steps: (1) obtain the ITA result from the argument relation and (2) merge adjacent ITA result tuples until a user specified size or error bound is satisfied. We begin by describing the merging of adjacent tuples and an error measure that is used to quantify the introduced error.

4.1 Merging Adjacent Tuples

The ITA result is always a sequential relation, which shall be preserved by allowing only adjacent tuples to be merged.

Definition 2 (Adjacent Tuples) Let $\mathbf{s}$ be a sequential relation with schema $S = (A_1, \dots, A_k, B_1, \dots, B_p, T)$ and grouping attributes $\mathbf{A} = \{A_1, \dots, A_k\}$. Two tuples $s_i, s_j \in \mathbf{s}$ are *adjacent*, $s_i \prec s_j$, iff the following holds:

(1) $s_i.\mathbf{A} = s_j.\mathbf{A}$,

(2) $s_i.t_e = s_j.t_b - 1$.

The first condition ensures that the two tuples are value-equivalent in the non-temporal attributes. The second condition requires that the tuples are immediately consecutive and not separated by a temporal gap.

Example 2 In the ITA result in Fig. 1(c) we have $s_1 \prec s_2 \prec s_3 \prec s_4 \prec s_5$. Tuples s_5 and s_6 are not adjacent, $s_5 \not\prec s_6$, since the *Proj*-values are different, violating the first condition. Similar, s_6, s_7 and s_1, s_3 are not adjacent since they are separated by a temporal gap and violate the second condition.

Definition 3 (Merge Operator) Let $\mathbf{s}$ be an ITA result relation with schema $S = (A_1, \dots, A_k, B_1, \dots, B_p, T)$, where $\mathbf{A} = \{A_1, \dots, A_k\}$ are the grouping attributes and $\mathbf{B} = \{B_1, \dots, B_p\}$ store the aggregate values. The *merge*, $\oplus$, of two adjacent tuples, $s_i, s_j \in \mathbf{s}$, $s_i \prec s_j$, is defined as

$$s_i \oplus s_j = (s_i.A_1, \dots, s_i.A_k, v_1, \dots, v_p, [s_i.t_b, s_j.t_e]),$$

where $v_d = \frac{|s_i.T| s_i.B_d + |s_j.T| s_j.B_d}{|s_i.T| + |s_j.T|}$ for $1 \leq d \leq p$.

The merge operator produces a new tuple from two ITA result tuples, i.e., $z = s_i \oplus s_j$. The values of the grouping attributes of z are identical to the ones of s_i (and s_j). The timestamp $z.T$ is the concatenation of the timestamps of s_i and s_j. Since the aggregate values of s_i and s_j hold at every time point in $s_i.T$ and $s_j.T$, respectively, the new aggregate values, $v_1, \dots, v_p$, are computed by averaging over the timestamps, i.e., v_d is the weighted average of $s_i.B_d$ and $s_j.B_d$ with the weights being the length of $s_i.T$ and $s_j.T$, respectively.

Example 3 Merging the two tuples $s_1 = (A, 800, [1,2])$ and $s_2 = (A, 600, [3,3])$ in Fig. 1(c) yields the result tuple $z_1 = s_1 \oplus s_2 = (A, 733.33, [1,3])$ in Fig. 1(d). The average salary is determined as $z_1.AvgSal = (2 \cdot 800 + 1 \cdot 600)/(2+1) = 733.33$.

To reduce the ITA result, $\mathbf{s}$, to a specific size, the merge operator is applied recursively. However, there is a *lower bound*, c_{min}, for the size of the reduced ITA result, which is determined by the difference between the cardinality of $\mathbf{s}$ and the number of adjacent tuple pairs that can be merged, i.e., $c_{min} = |\mathbf{s}| - |\{(s_i, s_j) | s_i, s_j \in \mathbf{s} \land s_i \prec s_j\}|$. In our running example, the ITA result contains seven tuples with four adjacent pairs, giving $c_{min} = 7 - 4 = 3$.

Next, we introduce a (nondeterministic) reduction function that reduces an ITA result relation to a given size c.

Definition 4 (Reduction Function) Let $\mathbf{s}$ be an ITA result relation, $s_i \prec s_j$ be two adjacent tuples in $\mathbf{s}$, and $c \geq c_{min}$ be a size constraint. The *reduction*, ρ, of relation $\mathbf{s}$ to size c is defined as

$$\rho(\mathbf{s}, c) = \begin{cases} \mathbf{s} & |\mathbf{s}| \leq c, \\ \rho(\mathbf{s} \setminus \{s_i, s_j\} \cup \{s_i \oplus s_j\}, c) & |\mathbf{s}| > c. \end{cases}$$

If the cardinality of $\mathbf{s}$ is smaller or equal to c, the reduction process terminates. Otherwise, two adjacent tuples, s_i and s_j, are substituted by the merged tuple $s_i \oplus s_j$. Notice the nondeterministic nature of ρ which allows any pair of adjacent tuples to be merged. We will be more specific about choosing tuples for merging later on.

Example 4 The ITA result relation in Fig. 1(c) is reduced to size $c = 4$ in three merging steps with $\rho(\mathbf{s}, 4)$. The reduced relation in Fig. 1(d) is obtained by merging tuples s_1, s_2 into z_1 and $s_3 \oplus (s_4 \oplus s_5)$ into z_2. Choosing different pairs of tuples for merging produces different results.

4.2 The Error Measure

Merging tuples introduces an error with respect to the ITA result, which we quantify using the following error measure.

Definition 5 (Error Measure) Let $\mathbf{s}, S, \mathbf{A}, \mathbf{B}$ be as in Def. 3, $\mathbf{z} = \rho(\mathbf{s}, \cdot)$ be a reduction of $\mathbf{s}$, and let for each $z \in \mathbf{z}$, $\mathbf{s}_z = \{s \mid s \in \mathbf{s} \land s.\mathbf{A} = z.\mathbf{A} \land s.T \subseteq z.T\}$ be the set of all ITA result tuples that are merged into z. For a set of positive weights, $w_1 > 0, \dots, w_p > 0$, the *error*, $SSE(\mathbf{s}, \mathbf{z})$, that is introduced by reducing $\mathbf{s}$ to $\mathbf{z}$ is

$$SSE(\mathbf{s}, \mathbf{z}) = \sum_{z \in \mathbf{z}} \sum_{s \in \mathbf{s}_z} \sum_{d=1}^{p} w_d^2 |s.T| (s.B_d - z.B_d)^2.$$

This is the well-known sum squared error, which is given as the total sum of the squared distance between the tuples in $\mathbf{s}$ and $\mathbf{z}$. More specifically, it computes for each tuple, $z \in \mathbf{z}$, the squared distance (over all aggregation results, $B_1, \dots, B_p$) between z and the ITA result tuples, $s \in \mathbf{s}_z$, that are merged to produce z. The weights w_d are used to leverage the impact of the different aggregation attributes. The choice of such weights is out of the scope of this paper; the interested reader is referred to Wettschereck et al. [29].

Example 5 Consider the merge of $s_1 = (A, 800, [1,2])$ and $s_2 = (A, 600, [3,3])$ in Fig. 1(c) to tuple $z = (A, 733.33, [1,3])$ in Fig. 1(d). With a weight of 1 for the only aggregation attribute *AvgSal*, the introduced error is $SSE(\mathbf{s},\mathbf{z}) = 1{\cdot}2{\cdot}(800-733.33)^2 + 1{\cdot}1{\cdot}(600-733.33)^2 = 26\,666.67$.

4.3 The PTA Operator

We provide two variants of the PTA operator. First, size-bounded PTA reduces the ITA relation to a user-specified size, while minimizing the introduced error. Second, error-bounded PTA reduces the size of the ITA result relation as much as possible, while maintaining the total introduced error below a given threshold.

Definition 6 (Size-Bounded PTA) Let $\mathbf{r}$ be a temporal relation with schema $R = (A_1, \dots, A_m, T)$, grouping attributes $\mathbf{A} = \{A_1, \dots, A_k\}$, and aggregate functions $\mathbf{F} = \{f_1/B_1, \dots, f_p/B_p\}$, and let $c \geq c_{min}$ be an application-specific size constraint. A relation $\mathbf{z}$ is the result of *size-bounded parsimonious temporal aggregation*, $\mathbf{z} = \mathscr{G}^{PTA}[\mathbf{A}, \mathbf{F}, c]\mathbf{r}$, iff

(1) $\mathbf{s} = \mathscr{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r}$,

(2) $\mathbf{z} = \rho(\mathbf{s}, c)$,

(3) $\nexists \mathbf{z}'(\mathbf{z}' = \rho(\mathbf{s}, c) \wedge SSE(\mathbf{s}, \mathbf{z}') < SSE(\mathbf{s}, \mathbf{z}))$.

Relation $\mathbf{s}$ is the ITA result, which is reduced to c tuples in the best possible way, that is, there is no better reduction, $\mathbf{z}'$, of $\mathbf{s}$ to $c' < c$ tuples that would introduce a smaller error. A PTA result is not necessarily unique. If different reductions to size c introduce the same minimal error, all of them represent valid PTA results.

Example 6 There are four different ways to reduce the ITA result in Fig. 1(c) to $c = 4$ tuples. Fig. 1(d) shows the best possible reduction with an introduced error of 49 166. Fig. 9 shows a different reduction, which has an error of 63 000.

Definition 7 (Error-Bounded PTA) Let $\mathbf{r}$, R, $\mathbf{A}$, and $\mathbf{F}$ be as in Def. 6 and ε, $0 \leq \varepsilon \leq 1$, be an application-specific error bound. Furthermore, let $SSE_{max} = SSE(\mathbf{s}, \rho(\mathbf{s}, c_{min}))$ denote the largest possible error. A relation $\mathbf{z}$ is the result of *error-bounded parsimonious temporal aggregation*, $\mathbf{z} = \mathscr{G}^{PTA}[\mathbf{A}, \mathbf{F}, \varepsilon]\mathbf{r}$ iff

(1) $\mathbf{s} = \mathscr{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r}$,

(2) $\exists c(\mathbf{z} = \rho(\mathbf{s}, c))$,

(3) $SSE(\mathbf{s}, \mathbf{z}) \leq \varepsilon{\cdot}SSE_{max}$,

(4) $\nexists \mathbf{z}', c'(\mathbf{z}' = \rho(\mathbf{s}, c') \wedge c' \leq c \wedge SSE(\mathbf{s}, \mathbf{z}') < SSE(\mathbf{s}, \mathbf{z}))$.

Relation $\mathbf{z}$ is a maximal reduction of $\mathbf{s}$ (to a size c) such that the introduced error is smaller or equal to ε multiplied by the largest possible error, which occurs when $\mathbf{s}$ is reduced to c_{min} tuples. Condition 4 ensures that no reduction to a smaller or the same number, $c' \leq c$, of tuples exists with a smaller error. Again, the result may not be unique.

Example 7 With an error threshold $\varepsilon = 1$ we obtain obviously the maximal reduction of the **proj** relation to three tuples. Allowing 2% error yields 4 result tuples as in Fig. 1(d).

5 PTA Evaluation Using Dynamic Programming

For the evaluation of PTA queries, ITA is computed first, followed by a reduction of the ITA result until a given size or error bound is satisfied. In this section we propose algorithms PTA_c and PTA_ε for the precise evaluation of size-bounded and error-bounded PTA, respectively. While any ITA algorithm can be used for the first step, we adopt a dynamic programming based approach to compute an optimal reduction of the ITA result. We further propose various optimization techniques to improve the basic DP scheme, such as computing the error in constant time and exploiting temporal gaps and aggregation groups to prune the search space.

5.1 Basic DP Scheme for Size-Bounded PTA

Let $\mathbf{s} = \{s_1, \dots, s_n\}$ be an ITA result relation sorted on the aggregation groups and, within each aggregation group, along the time line. Then each pair of consecutive tuples that are non-adjacent, $s_i \not\prec s_{i+1}$, marks a boundary (temporal gap or change of aggregation group) that cannot be crossed during the merging process.

Example 8 Consider the ITA result in Fig. 1(c), which is sorted first by the *Proj* attribute and, within each group, in chronological order. It contains two boundaries, namely s_5, s_6 since the two tuples belong to different aggregation groups, and s_6, s_7 since the two tuples are separated by a temporal gap.

Let $\mathbf{s}_j = \{s_1, \dots, s_j\}$ denote the first j tuples in $\mathbf{s}$ and $\mathbf{s} \setminus \mathbf{s}_j = \{s_{j+1}, \dots, s_n\}$ the rest. Then the reduction of $\mathbf{s}$ to c tuples, $\rho(\mathbf{s}, c)$, can be defined recursively as follows: find a reduction $\rho(\mathbf{s}_j, c-1)$ for some *split point* j and merge the remaining tuples into one, i.e., $\rho(\mathbf{s} \setminus \mathbf{s}_j, 1)$. For the reduction to be optimal, the sum of errors introduced on both sides of j must be minimized at each recursive step. To avoid that non-adjacent tuples are merged, we set the error of merging non-adjacent tuples to infinity. Thus, merging altogether any subset $\mathbf{s}' \subseteq \mathbf{s}$ yields an infinite error if $\mathbf{s}'$ contains at least one pair of non-adjacent tuples, $s_i \not\prec s_{i+1}$.

Example 9 Fig. 3 illustrates the four options for the split point, j. For instance, for $j = 3$, the solution is to find an optimal reduction $\rho(\mathbf{s}_3, 3)$ and to merge s_4, s_5, s_6, s_7 into one tuple, which yields an infinite error since $s_5 \not\prec s_6 \not\prec s_7$. The only split point with an error different from ∞ is $j = 6$.

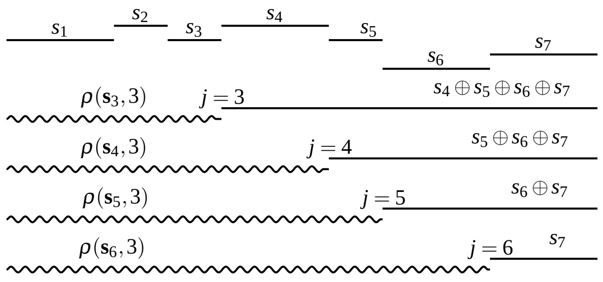


Fig. 3 Four possible ways to reduce the ITA result to four tuples.

To find an optimal reduction, we propose a dynamic programming technique that constructs an *error matrix*, $\mathbf{E}_{c\times n}$, with c rows and n columns. A cell (k,i) represents the smallest error of reducing $\mathbf{s}_i$ to k tuples. The matrix is filled incrementally in each step using the values that have been computed in the previous steps, i.e., $\mathbf{E}_{k,i} =$

$$\begin{cases} \min\limits_{k-1\le j<i} \{\mathbf{E}_{k-1,j} + \\ \qquad SSE(\mathbf{s}_i \backslash \mathbf{s}_j, \rho(\mathbf{s}_i \backslash \mathbf{s}_j, 1))\} & k > 1, \\ SSE(\mathbf{s}_i, \rho(\mathbf{s}_i, 1)) & k=1 \wedge s_1 \prec \ldots \prec s_i, \\ \infty & k=1 \wedge \neg(s_1 \prec \ldots \prec s_i). \end{cases}$$

The matrix is filled row-wise for all $k = 1,\ldots,c$, and, for any fixed k, in increasing order of i for $i = 1,\ldots,|\mathbf{s}|$. At each step k, the values computed in step $k-1$ are used. At the end, the value $\mathbf{E}_{c,n}$ contains the error introduced by an optimal reduction of relation $\mathbf{s}$ to c tuples.

Example 10 Figure 4 shows the error matrix that is constructed when reducing the ITA result in our running example to size 4. The matrix is filled starting from row $k = 1$. To fill the second row, the data from row 1 are used, etc. Eventually, cell $(4,7)$ contains the error of the optimal reduction.

	$i=1$	2	3	4	5	6	7
$k=1$	0	26666	67500	208333	269285	∞	∞
2	–	0	5000	41666	49166	269285	∞
3	–	–	0	5000	6666	49166	269285
4	–	–	–	0	1666	6666	$\boxed{49166}$

Fig. 4 Error matrix $\mathbf{E}$.

In order to construct the reduced relation, we maintain a *split point matrix*, $\mathbf{J}_{c\times n}$. A cell (k,i) in the matrix stores the value of j that led to the minimal error value when computing $\mathbf{E}_{k,i}$. Consequently the cell (c,n) stores the first split point j that tells us where to split $\mathbf{s}$ in order to construct the final result. The tuples $s_{j+1},\ldots,s_n$ are then merged into a single one, whereas tuples $s_1,\ldots,s_j$ are merged into $c-1$ tuples, following the next split point that is stored in the cell $(c-1,j)$, etc.

Example 11 Figure 5 shows the split point matrix in our running example. The split points of the optimal reduction are framed. The first split point is $j = 6$, the value of cell $(4,7)$. We generate the result tuple $z_4 = s_7$ and proceed to reduce $\mathbf{s}_6$ to size 3 by taking the value of cell $(3,6)$ as the next split point. We generate the result tuple $z_3 = s_6$. Then we proceed to reduce $\mathbf{s}_5$ to 2 tuples, obtaining $z_2 = s_3 \oplus s_4 \oplus s_5$. Finally, we reduce $\mathbf{s}_2$ to size 1, yielding $z_1 = s_1 \oplus s_2$ (the last split point in cell $(2,1)$ is 0).

	$i=1$	2	3	4	5	6	7
$k=1$	0	$\boxed{0}$	0	0	0	0	0
2	–	1	1	2	$\boxed{2}$	5	0
3	–	–	2	3	3	$\boxed{5}$	6
4	–	–	–	3	3	5	$\boxed{6}$

Fig. 5 Split point matrix $\mathbf{J}$.

5.2 Efficient Computation of the Error

The DP scheme frequently needs to compute the error that is introduced when a set of adjacent tuples is merged. Jagadish et al. [11] introduce a technique to calculate the error for one-dimensional data in constant time. We extend their approach for multi-dimensional data.

Let $\mathbf{s}$ be an ITA result relation with aggregation attributes $\mathbf{B} = \{B_1,\ldots,B_p\}$. The additional information that is required for an efficient computation of the error is stored in two matrices $\mathbf{S}_{p\times|\mathbf{s}|}, \mathbf{SS}_{p\times|\mathbf{s}|}$ and a vector $\mathbf{L}_{|\mathbf{s}|}$, which are defined as follows:

$$\mathbf{S}_{d,i} = \sum_{j=1}^{i} |s_j.T| s_j.B_d,$$

$$\mathbf{SS}_{d,i} = \sum_{j=1}^{i} |s_j.T| s_j.B_d^2,$$

$$\mathbf{L}_i = \sum_{j=1}^{i} |s_j.T|.$$

$\mathbf{S}_{d,i}$ is the sum of the B_d values over all tuples from s_1 to s_i, $\mathbf{SS}$ is the sum of the squares of the B_d values, and $\mathbf{L}$ is the sum of the lengths of the timestamps. Observe that precomputing this information does not introduce any additional overhead since the ITA algorithm can fill the matrices while producing the output.

Using such information, the error of merging a set of ITA result altogether can be computed in $O(p)$ time, where p is the number of aggregation attributes in the ITA result. This is shown in the following proposition.

Proposition 1 *Let $\mathbf{s}_z = \{s_i, s_{i+1}, \dots, s_j\} \subseteq \mathbf{s}$ such that $s_i \prec \dots \prec s_j$. The error that is introduced by merging $\mathbf{s}_z$ into one tuple, z, can be computed as*

$$SSE(\mathbf{s}_z, \{z\}) = \sum_{d=1}^{p} w_d^2 \left[\mathbf{SS}_{d,j} - \mathbf{SS}_{d,i-1} - \frac{(\mathbf{S}_{d,j} - \mathbf{S}_{d,i-1})^2}{\mathbf{L}_j - \mathbf{L}_{i-1}} \right].$$

Proof From Def. 3 of the merge operator we have that $z.B_d = \frac{1}{|z.T|} \sum_{s \in \mathbf{s}_z} |s.T| s.B_d$. We rewrite the error equation in Def. 5 as follows:

$$\begin{aligned} SSE(\mathbf{s}_z, \{z\}) &= \sum_{d=1}^{p} w_d^2 \Bigg[\sum_{s \in \mathbf{s}_z} |s.T| s.B_d^2 - \\ &\qquad 2z.B_d \underbrace{\sum_{s \in \mathbf{s}_z} |s.T| s.B_d}_{=|z.T| z.B_d} + z.B_d^2 \underbrace{\sum_{s \in \mathbf{s}_z} |s.T|}_{=|z.T|} \Bigg] \\ &= \sum_{d=1}^{p} w_d^2 \left[\sum_{s \in \mathbf{s}_z} |s.T| s.B_d^2 - |z.T| z.B_d^2 \right] \\ &= \sum_{d=1}^{p} w_d^2 \left[\sum_{s \in \mathbf{s}_z} |s.T| s.B_d^2 - \frac{(\sum_{s \in \mathbf{s}_z} |s.T| s.B_d)^2}{|z.T|} \right] \\ &= \sum_{d=1}^{p} w_d^2 \left[\mathbf{SS}_{d,j} - \mathbf{SS}_{d,i-1} - \frac{(\mathbf{S}_{d,j} - \mathbf{S}_{d,i-1})^2}{\mathbf{L}_j - \mathbf{L}_{i-1}} \right]. \end{aligned}$$

$\square$

Example 12 For the ITA result in Fig. 1(c) the matrices and vectors are given as follows:

$$\begin{aligned} \mathbf{S} &= \langle \quad 1600, \quad 2200, \quad 2700, \quad 3400, \dots \rangle, \\ \mathbf{SS} &= \langle 1280000, 1640000, 1890000, 2135000, \dots \rangle, \\ \mathbf{L} &= \langle \quad 2, \quad 3, \quad 4, \quad 6, \dots \rangle. \end{aligned}$$

Using this information, the error of merging the tuples $\{s_2, s_3\}$ into a tuple z is computed as $SSE(\{s_2, s_3\}, \{z\}) = 1890000 - 1280000 - \frac{(2700-1600)^2}{4-2} = 5000$.

5.3 Pruning the Search Space of the DP Scheme

Recall that filling the error matrix $\mathbf{E}$ involves computing the value of each cell (k,i) for all $k = 1, \dots, c$ and $i = 1, \dots, n$ using the above dynamic programming equation. This leads to an algorithm whose performance depends quadratically on the input size and linearly on the output size. In this section we introduce bounds for the variables i and j (in the equation) to improve the performance of the algorithm. Bounding variable i allows us to avoid the computation of some $\mathbf{E}_{k,i}$ if that would anyway evaluate to infinity. Otherwise, we speed up the evaluation of $\mathbf{E}_{k,i} = \min_{k-1 \le j < i} \{\mathbf{E}_{k-1,j} + SSE(\mathbf{s}_i \setminus \mathbf{s}_j, \rho(\mathbf{s}_i \setminus \mathbf{s}_j, 1))\}$ by reducing the value range of the variable j. The bounds depend on the positions of the non-adjacent tuple pairs in the sorted input relation $\mathbf{s}$. Thus, the performance improvements are data-dependent.

Let $\mathbf{G}$ be a vector that stores the positions of the non-adjacent tuple pairs in the sorted input relation, $\mathbf{s}$, i.e., $\mathbf{G}_m = l$ if $s_l, s_{l+1} \in \mathbf{s}$, $s_l \not\prec s_{l+1}$, is the m-th pair of non-adjacent tuples. We use the information in $\mathbf{G}$ to compute the bounds of i and j variables.

Example 13 For the ITA result of the running example we have $\mathbf{G} = \langle 5, 6 \rangle$, which is illustrated in Fig. 6. The first pair of non-adjacent tuples is $s_5 \not\prec s_6$. The second pair is $s_6 \not\prec s_7$.

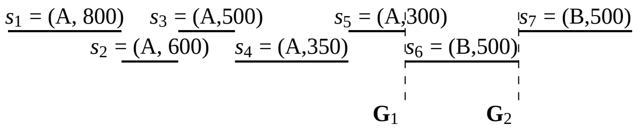


Fig. 6 The vector of gaps, $\mathbf{G}$.

First, we determine an upper bound, i_{max}, for the variable i under which $\mathbf{E}_{k,i}$ does not evaluate to infinity. Intuitively, if the number of non-adjacent tuple pairs in $\mathbf{s}_i$ is greater than k, then merging across gaps is unavoidable, and, we are sure that the error $\mathbf{E}_{k,i}$ is infinite. As long as $k \le |\mathbf{G}|$, the value $\mathbf{G}_k$ tells us the position of the k-th non-adjacent tuple pair. Consequently, the subset $\mathbf{s}_i = \{s_1, \dots, s_{\mathbf{G}_k}\} \subseteq \mathbf{s}$ has $k-1$ non-adjacent tuple pairs and is the maximal subset that can be reduced to size k. Therefore, $i_{max} = \mathbf{G}_k$ and for all $i > i_{max}$ we have $\mathbf{E}_{k,i} = \infty$. When $k > |\mathbf{G}|$, the rule may no longer be applied and we set i_{max} equal to the size of the input relation, $i_{max} = |\mathbf{s}|$. The more non-adjacent tuple pairs are present in the relation, the more advantageous is this upper bound to speed up the evaluation.

Example 14 Consider the computation of $\mathbf{E}_{1,i}$ using the vector $\mathbf{G} = \langle 5, 6 \rangle$ shown in Fig. 6. The value $\mathbf{G}_1 = 5$ indicates that at most the first five tuples, $\mathbf{s}_5 = \{s_1, \dots, s_5\}$, can be merged into one tuple without crossing a gap and inducing an infinite error. Therefore, given $k = 1$, the upper bound for i is $i_{max} = \mathbf{G}_1 = 5$; for all $i > 5$ we have $\mathbf{E}_{1,i} = \infty$. Given $k = 2$ the upper bound for i is $i_{max} = \mathbf{G}_2 = 6$. For all greater values of k the rule does not apply and i cannot be upper-bounded.

Second, whenever $E_{k,i}$ must be evaluated, we can determine a lower bound, j_{min}, for the variable j. The recursion formula for $\mathbf{E}_{k,i}$ determines the error of merging the tuples $\mathbf{s}_i \setminus \mathbf{s}_j$ into one tuple, which will be infinite if $\mathbf{s}_i \setminus \mathbf{s}_j$ contains at least one non-adjacent tuple pair. This is the case if j is smaller than the position of the right-most non-adjacent tuple pair in $\mathbf{s}_i$, if such a pair exists. The lower bound for j is therefore the position of the right-most non-adjacent tuple pair, i.e., $j_{min} = \max\{\mathbf{G}_l \mid \mathbf{G}_l < i \wedge l = 1,\dots,|\mathbf{G}|\}$. If $\mathbf{s}_i$ contains no gaps, we set $j_{min} = k-1$. Hence, to evaluate $\mathbf{E}_{k,i}$ it is enough to loop only over $j_{min} \le j < i$. To efficiently determine j_{min}, we use binary search over $\mathbf{G}$. If there are no gaps in $\mathbf{s}_i$, the search does not return any result and we set $j_{min} = k-1$. When $j_{min} = G_{k-1}$, the subset $\mathbf{s}_i$ has exactly k gaps and the only choice to split $\mathbf{s}_i$ is at $j = j_{min}$.

Example 15 To compute $\mathbf{E}_{3,6}$, the basic DP scheme evaluates the SSE of merging the tuples $\mathbf{s}_6 \setminus \mathbf{s}_j$ for $j = 2,\dots,5$. The right-most non-adjacent pair in $\mathbf{s}_6$ is $j_{min} = \mathbf{G}_1 = 5$. Therefore, only for $j = 5$ the error is different from ∞; the error computation for $j = 2,\dots,4$ can safely be pruned.

5.4 The Size-Bounded PTA Algorithm

Figure 7 shows algorithm PTA_c for the precise evaluation of size-bounded PTA queries using the above DP scheme. First, the ITA result, $\mathbf{s}$, over the input relation $\mathbf{r}$ is computed using any ITA algorithm. (We assume $\mathbf{s}$ to be sorted by the grouping attributes $\mathbf{A}$ and, within each group, in chronological order; if not, an additional sorting step is required.) Next, the vectors $\mathbf{G}$, $\mathbf{L}$ and matrices $\mathbf{S}$, $\mathbf{SS}$ that are needed for the error computation are initialized. Notice that this initialization could be pushed into the ITA algorithm to avoid an additional scan of $\mathbf{s}$. Next, the error, $\mathbf{E}$, and split point, $\mathbf{J}$, matrices are initialized. The following loop fills these matrices by implementing the DP scheme together with the performance improvements described above. For each matrix row, k, we iterate over columns, i, computing $\mathbf{E}_{k,i}$. The upper bound for i is obtained from the gap vector $\mathbf{G}$. When $k = 1$, we implement the first condition in the scheme, and the evaluation of $\mathbf{E}_{k,i}$ is straightforward. The following lines implement the second condition. When the number of non-adjacent pairs in the subset $\mathbf{s}_i$ equals to k, the only possible split point is at $j = j_{min}$. In all other cases, an iteration over j is necessary. We lower-bound the variable j, that is, j must be greater than the index of the right-most non-adjacent pair in the subset $\mathbf{s}_i$. Recall that $\mathbf{E}_{k,i}$ has been initialized to infinity. By iterating over j in decreasing order, we choose any smaller value. It has been shown by Jagadish et al. [11] that j should be iterated in decreasing order, i.e., from $i-1$ towards j_{min}. Since the value of err_2 increases with each iteration, the loop can be safely broken when e_2 alone exceeds the smallest error $\mathbf{E}_{k,i}$ found so far. The final while loop computes the output using the split point matrix $\mathbf{J}$ as described before.

Example 16 The evaluation of PTA_c over the **proj** relation starts with the computation of the ITA result $\mathbf{s}$. The tuples are enumerated from 1 to 7 as in Fig. 1(c). Next, $\mathbf{E}_{1,i}$ is computed for all $i = 1,\dots,5$. The values $\mathbf{E}_{1,6}$ and $\mathbf{E}_{1,7}$ are infinite and their evaluation will be avoided with the help of the upper bound i_{max}. Similarly, we compute $\mathbf{E}_{2,i}$ for all $i = 2,\dots,6$ and avoid the evaluation of $\mathbf{E}_{7,2}$. When $k = 2$ and i is between 2 and 5, the loop over j ranges between 1 and i. However, when i is 6, the value of j is fixed at 5. This way all the remaining errors are computed until $k = 4$ and $i = 7$, and the final output relation shown in Fig. 1(d) is produced.

```
Algorithm: $PTA_c(\mathbf{r},\mathbf{A},\mathbf{F},c)$
$\mathbf{s} \leftarrow \mathscr{G}^{ITA}[\mathbf{A},\mathbf{F}]\mathbf{r}$;
Initialize $\mathbf{G},\mathbf{L},\mathbf{S},\mathbf{SS}$;
Initialize $\mathbf{E},\mathbf{J}$ to $\infty$ and 0, respectively;
for $k = 1,\dots,c$ do
    if $k \le |\mathbf{G}|$ then $i_{max} = \mathbf{G}_k$ else $i_{max} = |\mathbf{s}|$;
    for $i = k,\dots,i_{max}$ do
        if $k = 1$ then
            $\mathbf{E}_{1,i} \leftarrow SSE(\mathbf{s}_i,\rho(\mathbf{s}_i,1))$;
            $\mathbf{J}_{1,i} \leftarrow 0$;
        else
            $j_{min} \leftarrow \max\{k-1,\mathbf{G}_l|\mathbf{G}_l < i \wedge l = 1,\dots,|\mathbf{G}|\}$;
            if $\mathbf{G}_{k-1} = j_{min}$ then
                $j \leftarrow j_{min}$;
                $\mathbf{E}_{k,i} \leftarrow \mathbf{E}_{k-1,j} + SSE(\mathbf{s}_i \setminus \mathbf{s}_j,\rho(\mathbf{s}_i \setminus \mathbf{s}_j,1))$;
                $\mathbf{J}_{k,i} \leftarrow j$;
            else
                for $j = i-1,\dots,j_{min}$ do
                    $err_1 \leftarrow \mathbf{E}_{k-1,j}$;
                    $err_2 \leftarrow SSE(\mathbf{s}_i \setminus \mathbf{s}_j,\rho(\mathbf{s}_i \setminus \mathbf{s}_j,1))$;
                    if $err_1 + err_2 < \mathbf{E}_{k,i}$ then
                        $\mathbf{E}_{k,i} \leftarrow err_1 + err_2$;
                        $\mathbf{J}_{k,i} \leftarrow j$;
                    if $err_2 > \mathbf{E}_{k,i}$ then break;
$\mathbf{z} \leftarrow \emptyset$, $n \leftarrow |\mathbf{s}|$;
while $c > 0$ do
    $j \leftarrow \mathbf{J}_{c,n}$;
    $\mathbf{z} \leftarrow \mathbf{z} \cup \{s_{j+1} \oplus \dots \oplus s_n\}$;
    $n \leftarrow j$; $c \leftarrow c-1$;
return $\mathbf{z}$;
```

Fig. 7 The PTA_c algorithm for size-bounded PTA.

The runtime complexity of PTA_c depends on the ITA algorithm and the merging step. We assume that ITA is computed by one of several algorithms that have been proposed in the past, e.g. [4,15,18]. Their average running time is $O(n\log n)$, where n is the size of the input relation. In the merging step we evaluate the error within three nested loops, one per variable k, i, and j. The first two perform at most c and n iterations, respectively. The maximum number of iterations in j equals to the size of the largest adjacent tuple subset in the ITA result, say q. The error evaluation takes $O(p)$ time for p aggregation functions, however, p is usually insignificantly small and can be regarded as a constant. This yields a runtime complexity of $O(cnq)$ for the merging step in the PTA_c algorithm. In the worst case, when the dataset has no temporal gaps or aggregation groups, $q = n$ and the complexity of PTA_c is $O(n^2c)$. The space complexity of the algorithm is $O(n^2)$ as the split point matrix, $\mathbf{J}$, must be kept in memory entirely. On the other hand, only the two most recent rows of the error matrix, **E**, are necessary.

5.5 The Error-Bounded PTA Algorithm

To answer error-bounded PTA queries, we use the same DP scheme as for the size-bounded algorithm. The DP solution computes all optimal reductions to $k = 1, 2, \ldots$ tuples in increasing order of k. As k increases, the error monotonically decreases. Thus, the relation with the smallest k that satisfies the error bound ε is the solution.

Figure 8 depicts the evaluation algorithm for error-bounded PTA, which is similar to the algorithm in Fig. 7. The variable E_{max} is set to the maximum non-infinite error, i.e., the error that would be introduced by merging all adjacent tuples together. This value can be computed together with the ITA result at no additional cost. In the main loop, k iterates from 1 to the cardinality of the ITA relation, where the error matrix, **E**, and the split point matrix, **J**, are computed. The loop is terminated when the error exceeds ε. The runtime complexity of this algorithm is the same as for PTA_c.

```
1  Algorithm: $PTA_\varepsilon(\mathbf{r}, \mathbf{A}, \mathbf{F}, \varepsilon)$
2  $\mathbf{s} \leftarrow \mathscr{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r}$;
3  $\mathbf{E}_{max} \leftarrow SSE(\mathbf{s}, \rho(\mathbf{s}, c_{min}))$;
4  Initialize $\mathbf{S}, \mathbf{SS}, \mathbf{L}, \mathbf{G}, \mathbf{E}, \mathbf{J}$;
5  for $k = 1, \ldots, |\mathbf{s}|$ do
6      Fill $\mathbf{E}, \mathbf{J}$ using lines 6–24 in Fig. 7;
7      if $\mathbf{E}[|\mathbf{s}|, k] \le \varepsilon \cdot \mathbf{E}_{max}$ then
8          $c \leftarrow k$;
9          break;
10 Build output $\mathbf{z}$ using lines 25–29 in Fig. 7;
11 return $\mathbf{z}$;
```

Fig. 8 The PTA_ε algorithm for error-bounded PTA.

6 PTA Evaluation Using Greedy Merging

The DP approach computes a precise result with minimal merging error, but incurs a relatively high computational cost for large relations. Many applications, however, would benefit from a quick and cheap computation of an approximate answer to PTA queries. In this section we present an alternative evaluation strategy, which greedily merges the most similar pairs of ITA result tuples until the size or error bound is satisfied. The additional error introduced by greedy merging is reasonably small and upper bounded. We introduce two novel algorithms, $gPTA_c$ and $gPTA_\varepsilon$, for the greedy evaluation of size- and error-bounded PTA queries, respectively. We show that greedy merging can commence as the ITA result tuples arrive, i.e., before the whole result is computed. In this way the memory requirement is reduced to $O(c + \beta)$, where c is the PTA result size and β is typically a small fraction of the ITA result; the runtime complexity for the merging step is $O(n \log(c + \beta))$.

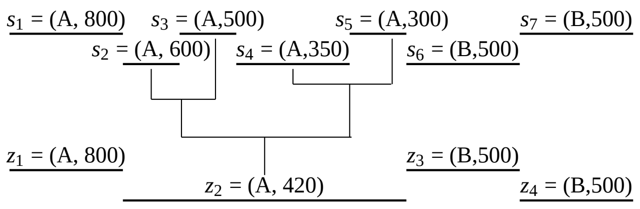


Fig. 9 The dendrogram of the greedy merging steps.

6.1 Greedy Merging Strategy

Let **s** be an ITA result relation. The *greedy merging strategy* (GMS) reduces **s** in an iterative manner until the size or error bound is satisfied. At each step, it operates on an intermediate result relation, choosing from it a pair of the most similar tuples for merging. Intuitively, the smaller the error of merging two tuples is, the more similar they are. By replacing the chosen pair with the newly merged tuple a reduced intermediate result is obtained. When several pairs of tuples are equally similar, any pair can be chosen. We elect to merge the pair with the smallest timestamp value. This choice, however, does not influence the total error introduced by the greedy merging process.

Example 17 Fig. 9 illustrates the greedy merging steps over the example ITA relation with size bound $c = 4$. The first tuples to be merged (i.e., the most similar ones) are s_4 and s_5 followed by s_2 and s_3. The two new tuples are then merged to produce the final result tuple z_2. The result of greedy merging differs from the precise PTA result in Fig. 1(d), where tuple s_2 is merged with s_1. The DP algorithm introduces an error of 49 166, while the error of the greedy approach is 63 000, yielding an error ratio of 1.28 between the two.

In order to apply GMS, the notion of similarity between tuple pairs needs to be precisely defined. Consider a sequential relation $\mathbf{s}'$ that is obtained from the initial relation, **s**, by applying the reduction operator. Merging a pair of adjacent tuples $s_i \in \mathbf{s}', s_j \in \mathbf{s}'$ leads to a new relation, say **z**. Then, the *dissimilarity* of the tuples s_i and s_j is the error introduced by the merge, i.e., $dsim(s_i, s_j) = SSE(\mathbf{s}, \mathbf{z}) - SSE(\mathbf{s}, \mathbf{s}')$. In order to determine the dissimilarity using this equation, the source relation **s** must be available, which is not practical if we want to start merging before the whole ITA result is computed. The following proposition shows that $dsim(s_i, s_j)$ can be determined by considering only the tuples s_i and s_j.

Proposition 2 *Let $\mathbf{s}$ be a sequential relation, $\mathbf{s}' = \rho(\mathbf{s}, k)$ be a reduction to size k, and $\mathbf{z}$ be obtained from $\mathbf{s}'$ by merging the tuples $s_i \in \mathbf{s}', s_j \in \mathbf{s}'$ to $s_i \oplus s_j = z \in \mathbf{z}$. The dissimilarity of the tuples s_i, s_j is $dsim(s_i, s_j) = SSE(\{s_i, s_j\}, \{z\})$.*

Proof Let $\mathbf{s}_i^* \subset \mathbf{s}$ be the tuples in $\mathbf{s}$ that make up s_i, i.e., for all $s \in \mathbf{s}_i^*$ we have $s[\mathbf{A}] = s_i[\mathbf{A}] \wedge s.T \subseteq s_i.T$. Let $\mathbf{s}_j^*$ be defined for s_j in a similar fashion. Then, $\mathbf{s}^* = \mathbf{s}_i^* \cup \mathbf{s}_j^*$ constitute z. Since the sets $\mathbf{s}' \setminus \{s_i, s_j\}$ and $\mathbf{z} \setminus \{z\}$ are identical we rewrite $dsim(s_i, s_j)$ as

$$\begin{aligned} dsim(s_i,s_j) &= SSE(\mathbf{s}^*,\{z\}) - SSE(\mathbf{s}^*,\{s_i,s_j\}) \\ &= SSE(\mathbf{s}_i^*,\{z\}) - SSE(\mathbf{s}_i^*,\{s_i\}) + & (a) \\ &\quad SSE(\mathbf{s}_j^*,\{z\}) - SSE(\mathbf{s}_j^*,\{s_j\}) & (b) \end{aligned}$$

The summation $(a) + (b)$ is possible because the tuples s_i, s_j are adjacent, i.e., they do not overlap. Recall, that according to the merge function, $s_i = \frac{1}{|s_i.T|} \sum_{s \in s_i^*} |s.T| s$. Using the definition of the error, we can rewrite equation (a) (and similarly (b)) as

$$\begin{aligned} (a) &= \sum_{s \in s_i^*} |s.T|(s-z)^2 - \sum_{s \in s_i^*} |s.T|(s-s_i)^2 \\ &= |s_i.T|(z^2 - 2z \cdot s_i + s_i^2) = SSE(\{z\},\{s_i\}). \end{aligned}$$

Since the square sum error is defined as a sum of individual errors and s_i, s_j are adjacent, we have

$$\begin{aligned} dsim(s_i,s_j) &= (a) + (b) = SSE(\{z\},\{s_i\}) + SSE(\{z\},\{s_j\}) \\ &= SSE(\{z\},\{s_i,s_j\}). \end{aligned}$$

□

In contrast to the DP approach, GMS does not necessarily compute an optimal ITA reduction. At every greedy merging step, there is a chance of making a sub-optimal decision. Therefore, the more merging steps are performed, the more additional error can be accumulated. The following theorem shows that the error ratio of the greedy and optimal solution is asymptotically upper-bounded by the logarithm of the number of merging steps. Experimentally we show that the greedy reduction is indeed very close to the optimal one.

Theorem 1 *Let $\mathbf{s}^n$ be a sequential relation of size n, $\mathbf{s}^c$ be a reduction of $\mathbf{s}^n$ to c tuples obtained using the greedy merging strategy, and $\mathbf{z}$ be an optimal reduction of $\mathbf{s}^n$ to c tuples obtained using PTA_c. The error ratio between the two solutions is*

$$\frac{SSE(\mathbf{s}^n,\mathbf{s}^c)}{SSE(\mathbf{s}^n,\mathbf{z})} \leq O(\log n).$$

Proof By the definition of the greedy merging strategy, the error introduced by the merge of two most similar tuples in $\mathbf{s}^k$ is

$$SSE(\mathbf{s}^k,\mathbf{s}^{k-1}) = \min_{\substack{s_i,s_j \in \mathbf{s}^k \\ s_i \prec s_j}} \{dsim(s_i,s_j)\}.$$

The minimum is upper-bounded by an average which is, in turn, upper-bounded by the total error, i.e.,

$$\begin{aligned} SSE(\mathbf{s}^k,\mathbf{s}^{k-1}) &\leq \frac{1}{k-1} \sum_{\substack{s_i,s_j \in \mathbf{s}^k \\ s_i \prec s_j}} dsim(s_i,s_j) \\ &\leq \frac{1}{k-1} C \cdot SSE(\mathbf{s}^n,\mathbf{z}). \end{aligned}$$

C is a constant whose minimal value may vary depending on k, yet there is a value that satisfies any k. According to Proposition 2, the error $SSE(\mathbf{s}^n,\mathbf{s}^c)$ made by the greedy algorithm is the sum of errors made at each intermediate merging step, i.e., $SSE(\mathbf{s}^n,\mathbf{s}^c) = \sum_{k=n}^{c+1} SSE(\mathbf{s}^k,\mathbf{s}^{k-1})$. Replacing the summand with the upper-bound we get $SSE(\mathbf{s}^n,\mathbf{s}^c) \leq C \cdot SSE(\mathbf{s}^n,\mathbf{z}) \sum_{k=n}^{c+1} \frac{1}{k-1}$, which leads to the error bound of the theorem, namely $\frac{SSE(\mathbf{s}^n,\mathbf{s}^c)}{SSE(\mathbf{s}^n,\mathbf{z})} \leq C \cdot \sum_{k=n}^{c+1} \frac{1}{k-1} \leq O(\log n)$. □

A straightforward implementation of the greedy merging strategy is to use a priority queue (e.g., a binary heap) to find the most similar tuple pairs. After inserting all ITA tuple pairs in the heap, the merging process starts, taking $O(n \log n)$ time and $O(n)$ space to compute any reduction of n ITA tuples. In the following we describe a more efficient implementation of the greedy merging strategy for size- and error-bounded PTA queries, which starts the merging process before the complete ITA result is available.

6.2 A Greedy Algorithm for Size-Bounded PTA

6.2.1 Basic Idea

We present the $gPTA_c$ algorithm for the evaluation of size-bounded PTA queries, which integrates the computation of the ITA result and greedy merging into one process. In a nutshell, $gPTA_c$ reads ITA result tuples as they become available and inserts them into a binary heap, which is used to efficiently identify the most similar tuple pair for merging. Whenever the heap contains more than c tuples, the algorithm attempts to merge the tuple at the top with its immediate predecessor, which requires some care. The tuples are only merged if GMS operating on the whole ITA relation would also choose to merge them. Such a situation can only be identified if the last two tuples in the heap are non-adjacent as stated by the following proposition. At any time during the computation the heap contains at most $c + \beta$ tuples. Since β is typically small, $gPTA_c$ improves over GMS in terms of running time and space efficiency.

Recall that the ITA result tuples come sorted along the aggregation groups, and, within each group, along the temporal dimension. We enumerate them from 1 to n, i.e., $\mathbf{s} = \{s_1, \dots, s_n\}$. Given a subset of $\mathbf{s}$, the following proposition specifies when GMS will merge the most similar tuples.

Proposition 3 *Let $\mathbf{s}_{j+1} = \{s_1, \dots, s_i, s_{i+1}, \dots, s_j, s_{j+1}\}$ be a part of the ITA result $\mathbf{s} = \{s_1, \dots, s_n\}$ and $s_i \prec s_{i+1}$ be the most similar pair of tuples in $\mathbf{s}_{j+1}$ for some $i \leq j$. The greedy merging strategy operating on $\mathbf{s}$ merges s_i and s_{i+1} if $j \geq c$ and $s_j \not\prec s_{j+1}$.*

Proof The GMS will not elect s_i, s_{i+1} for merging as long as there are more similar pairs in $\mathbf{s}$. Since s_i, s_{i+1} are most similar in $\mathbf{s}_{j+1}$, all pairs which are more similar than s_i, s_{i+1} (if any) must be in $\mathbf{s}_n \setminus \mathbf{s}_j$. Therefore, leaving s_i, s_{i+1} intact, the smallest intermediate relation produced by GMS is $\mathbf{s}' = \{s_1, \dots, s_i, s_{i+1}, \dots, s_j, s_{j+1} \oplus \dots \oplus s_n\}$. Since $|\mathbf{s}'| \geq |\mathbf{s}_j| > c$, more merging steps are needed to reduce $\mathbf{s}'$ to size c. Since the two tuples $s_j, s_{j+1} \oplus \dots \oplus s_n$ are not adjacent, the most similar pair in $\mathbf{s}'$ to be merged next can only be s_i, s_{i+1}. □

Example 18 Consider reading the first five tuples, $\mathbf{s}_5 = \{s_1, \dots, s_5\}$, of the ITA result in Fig. 9, and let the size bound for the PTA result be $c = 4$. The tuples $s_4 \prec s_5$ are the most similar ones, yet they cannot be merged since the tuple to be read next, s_6, might form an even more similar pair with s_5. Therefore, we read ahead an get $\mathbf{s}_6 = \{s_1, \dots, s_6\}$. Since $s_5 \not\prec s_6$, the GMS has to make a merge in the first five tuples, independent of the tuples that will follow.

Proposition 3 provides a criterion to perform early merging, yet guaranteeing the same result as GMS, namely: if more than c tuples are in the heap and the last tuple pair is not adjacent. When temporal gaps are rare or the aggregation groups are few, a large portion of the ITA result (in the worst case the whole result relation) may be inserted into the heap before a non-adjacent pair arrives. To avoid the heap growing much beyond size c, we propose a heuristic to determine whether the currently most similar tuple pair, s_i, s_{i+1}, would also be merged by GMS. Suppose that s_j is the last tuple in the heap and there is a more similar pair, s_k, s_{k+1}, in the ITA result that is connected to s_{i+1} by a sequence of adjacent tuples, but has not yet been inserted into the heap, i.e., $\mathbf{s}_{k+1} = \{s_1, \dots, s_i, s_{i+1}, \dots, s_j, \dots, s_k, s_{k+1}\}$, where $s_{i+1} \prec \dots \prec s_{k+1}$. GMS would first merge s_k, s_{k+1}. Since the merge result, $s_k \oplus s_{k+1}$, might be more similar to s_{k-1} than s_i, s_{i+1} are, the new tuple is next merged with s_{k-1}. This might propagate back, and s_{i+1} may potentially become more similar to its new successor (which is the result of several merging steps) than to s_i. In such a situation, merging s_i, s_{i+1} would be a mistake, leading to a result that is likely to be different from the GMS result. However, the more tuples follow the current merge candidate, s_i, s_{i+1}, the lower is the probability that the similarity of s_i, s_{i+1} will be influenced by merging subsequent tuples. Therefore, to keep the heap size small we use a parameter, δ, to specify the minimum number of adjacent tuples that have to follow the merge candidate for it to be merged. We will show experimentally that with $\delta = 1$ the difference between $gPTA_c$ and GMS is negligible, yet space and performance gain is significant.

Observe that $c + \delta$ is essentially the lower bound of the heap size. In the worst yet unlikely case the heap size will be equal to the ITA result size. The higher is the value of δ, the closer is the final result to that of GMS. When $\delta = \infty$, $gPTA_c$ and GMS produce the same output as shown by Theorem 2 below.

6.2.2 Heap Data Structure

We use a binary heap to avoid scanning the entire intermediate relation in search of the most similar pair of tuples. Given a relation $\mathbf{s}$, we represent a tuple $s \in \mathbf{s}$ as a node N that records the following information: the sequence number of the tuple ($N.id$), the tuple itself (s), a pointer to the previous (in chronological order) node (*prev*), a pointer to the next node (*next*), and the error that would be introduced by merging s with the previous tuple (*key*), i.e., $N.key = SSE(\mathbf{s}, \{N.s \oplus N.prev.s\})$. The key is set to ∞ if N and $N.prev$ represent non-adjacent tuples or s is the first tuple.

We define the following operations on the heap. INSERT creates a new node for a tuple and inserts it into the heap; this includes also the computation of the *key* value. PEEK returns the top node, N, but does not remove it from the heap. MERGE removes the top node, N, off the heap and merges the tuple $N.s$ into the preceding node, $P = N.prev$, yielding $P.s = P.s \oplus N.s$. The pointers $P.next$ and $N.next.prev$ are updated, the key values of $N.prev$ and of $N.next$ are recomputed, and the heap structure is updated. The field $P.id$ remains unchanged.

Example 19 Figure 10 depicts a binary heap. Solid lines represent parent-child relationships, dashed lines indicate *prev* and *next* links. In Fig. 10(a) the heap contains the whole ITA result. The key of s_1 is infinite since s_1 is the first tuple, whereas the key of s_6 is infinite because s_5 and s_6 are not adjacent. The most similar tuple pair is s_4, s_5. Thus, the node on the peak represents s_5 with the key value 1 667, which is the error of merging s_4 and s_5. Figure 10(b) shows the heap after performing one merge. Node s_5 is merged into node s_4, which now contains $s_4 \oplus s_5$. The key value of $s_4 \oplus s_5$ and s_6 are re-evaluated, and the *next* pointer of $s_4 \oplus s_5$ and the *prev* pointer of s_6 are updated. The new peak node is s_3, thus s_2 and s_3 will be merged next.

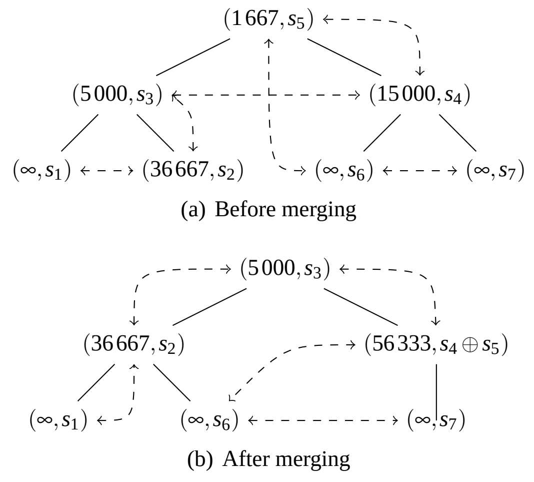


Fig. 10 Binary heap used in the *gPTA*$_c$ algorithm.

6.2.3 *gPTA*$_c$ *Algorithm*

Figure 11 shows the algorithm *gPTA*$_c$ for the greedy evaluation of size-bounded PTA. It takes as input parameters the size bound, c, and δ. The first step is to initialize an empty heap, the ITA operator that produces a sorted output, and three variables, *LastGapId*, *BG*, and *AG*. Variable *LastGapId* stores the sequence number of the last seen non-adjacent tuple pair, i.e., it is equal to the $N.id$ of the last seen node with key value ∞. Variables *BG* and *AG* store the number of tuples that preceed and, respectively, succeed the last non-adjacent node, *LastGapId*. Trivial modifications to the ITA algorithm proposed by Böhlen et al. [4] are necessary to allow processing the tuples one by one as they become available from the operator.

For each incoming ITA tuple a new node, N, is created and inserted into the heap. If $N.key = \infty$, the node represents a non-adjacent pair, variable *LastGapId* is set to the sequence number of this node and *BG* is increased by the value of *AG*. Otherwise, *AG* is incremented by 1.

Example 20 Assume that the first six tuples of the ITA result in Fig. 9 are read. Since five tuples precede the latest gap, $BG = 5$. Only one tuple follows the gap, thus $AG = 1$. Finally, $LastGapId = 6$ indicates the non-adjacent tuple that follows immediately after the gap.

When the size of the heap exceeds the PTA size bound, c, the merging process starts with the second while loop. First, the node with the smallest key value is read from the heap. Then we check whether merging can take place using Proposition 3. If the condition evaluates to true, we proceed with merging. Otherwise, we make use of the heuristic and only check whether the node has at least δ adjacent successors. The higher is δ, the less likely it is that the choice to merge is different from that of GMS and that *gPTA*$_c$ will deviate from GMS result. When $\delta = \infty$ the effect of the parameter is eliminated and merging will only happen when non-adjacent tuple pairs are discovered. In that case the algorithm is guaranteed to produce the same result as GMS. If none of the two conditions is true, merging is not possible. We break the cycle and wait for more tuples to be inserted into the heap. Finally, if the whole ITA result is read and the heap still does not satisfy the size bound, we merge the the most similar tuples until $|H| = c$.

```
Algorithm: $gPTA_c(\mathbf{r}, \mathbf{A}, \mathbf{F}, \delta, c)$
$H \leftarrow$ new empty heap;
Initialize the ITA operator with $\mathbf{F}$, $\mathbf{A}$, and $\mathbf{r}$;
$LastGapId \leftarrow 0$; $BG \leftarrow 0$; $AG \leftarrow 0$;
while $s_i \leftarrow$ next tuple from $\mathscr{G}^{ITA}[\mathbf{A},\mathbf{F}]\mathbf{r}$ do
    $N \leftarrow$ INSERT($s_i$);
    if $N.key = \infty$ then
        $LastGapId \leftarrow N.id$;
        $BG \leftarrow BG + AG$;
        $AG \leftarrow 1$;
    else
        $AG \leftarrow AG + 1$;
    while $|H| > c$ do
        $N \leftarrow$ PEEK();
        if $N.id < LastGapId \wedge BG \geq c$ then
            $BG \leftarrow BG - 1$;
            MERGE();
        else if $N.id > LastGapId \wedge N$ has $\delta$ adjacent
        successors then
            $AG \leftarrow AG - 1$;
            MERGE();
        else
            break;
while $|H| > c > c_{min}$ do
    MERGE();
return $H$;
```

Fig. 11 Greedy algorithm, *gPTA*$_c$, for size-bounded PTA.

Example 21 Consider running *gPTA*$_c$ over the **proj** relation with $c = 3$ and $\delta = 1$. Figure 12 depicts each intermediate state of the heap. The ITA tuples come sorted: first the group "A" and only then the group "B". In (a) the heap contains the first four ITA tuples. The merging process can start as the most similar pair, s_2 and s_3, has 1 successor. In (b) $s_2 \oplus s_3$ has been created to keep the heap size equal to 3. Next, tuple s_5 is inserted leading to (c) where the tuple at the top of the heap is s_5. Even though the size of the heap exceeds c, the merge cannot happen since s_5 does not have $\delta = 1$ successors. Observe that tuple s_5 may happen to be more similar to s_6 (which is not known yet) and, therefore should be merged with it. When tuple s_6 arrives (d), it becomes clear that merging is possible. The heap now contains five tuples. The algorithm repeatedly merges adjacent tuples (e,f) until the size constraint is satisfied again. Specifically, s_4 is merged to s_5

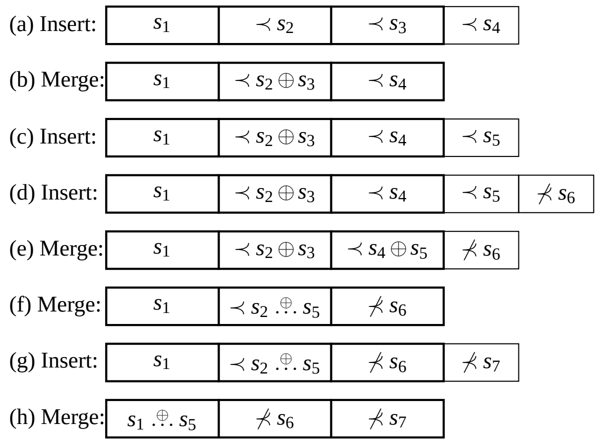

(a) Insert: s_1 | $\prec s_2$ | $\prec s_3$ | $\prec s_4$
(b) Merge: s_1 | $\prec s_2 \oplus s_3$ | $\prec s_4$
(c) Insert: s_1 | $\prec s_2 \oplus s_3$ | $\prec s_4$ | $\prec s_5$
(d) Insert: s_1 | $\prec s_2 \oplus s_3$ | $\prec s_4$ | $\prec s_5$ | $\not\prec s_6$
(e) Merge: s_1 | $\prec s_2 \oplus s_3$ | $\prec s_4 \oplus s_5$ | $\not\prec s_6$
(f) Merge: s_1 | $\prec s_2 \overset{\oplus}{\dots} s_5$ | $\not\prec s_6$
(g) Insert: s_1 | $\prec s_2 \overset{\oplus}{\dots} s_5$ | $\not\prec s_6$ | $\not\prec s_7$
(h) Merge: $s_1 \overset{\oplus}{\dots} s_5$ | $\not\prec s_6$ | $\not\prec s_7$

Fig. 12 Reducing an ITA result to three tuples with *$gPTA_c$*.

and the result is then merged with $s_2 \oplus s_3$. Finally, with the arrival of tuple s_7 (g) the final result is computed as shown in (h). Over the whole process the heap contained at most five tuples, whereas seven ITA tuples have been processed.

Theorem 2 *The output of the $gPTA_c$ algorithm with $\delta = \infty$ is identical to that of GMS.*

Proof The proof follows from Proposition 3. *$gPTA_c$* merges the same tuple pairs that GMS would merge. □

The complexity of *$gPTA_c$* depends on the ITA algorithm. Assume that the latter takes T time to produce a result relation of size n. In addition, it uses S space for internal structures. Then, $gPTA_c$ requires $O(T + n\log(c+\beta))$ time and $O(S+c+\beta)$ space (assuming that the heap operations take logarithmic time). In the worst case $c+\beta = n$, however for the majority of datasets we expect $c+\beta$ to be much smaller than n.

6.3 A Greedy Algorithm for Error-Bounded PTA

The algorithm, *$gPTA_\varepsilon$*, for the greedy evaluation of error-bounded PTA queries follows the same intuition as its size-bounded counterpart. As ITA result tuples are produced, it starts to merge tuples as early as possible and tries to merge as many tuples as possible before exceeding the error threshold ε. The major difference is on how to determine tuple pairs that would also be merged by GMS. For that we need to know the size, n, of the ITA result, $\mathbf{s}$, and the maximal error, $E_{max} = SSE(\mathbf{s}, \rho(\mathbf{s}, c_{min}))$, of reducing $\mathbf{s}$ to the smallest possible size, c_{min}. Since we do not wait until the ITA result is completed, these values have to be estimated. The ITA relation can be safely estimated to contain twice as many tuples as the argument relation. In order to estimate the total error we have to obtain a good sample of the ITA result. Using these two values, the following preposition shows how to identify tuple pairs in a subset of the ITA result that will be merged by GMS.

Proposition 4 *Let $\mathbf{s}_{j+1} = \{s_1,\dots,s_i,s_{i+1},\dots,s_j,s_{j+1}\}$ be a subset of the ITA result relation, $\mathbf{s} = \{s_1,\dots,s_n\}$, $s_i \prec s_{i+1}$ be the most similar pair of tuples in $\mathbf{s}_{j+1}$ for some $i \leq j$, and $E_{max} = SSE(\mathbf{s}, \rho(\mathbf{s},1))$ be the maximum total error. The greedy merging strategy operating on $\mathbf{s}$ will merge s_i and s_{i+1} if $dsim(s_i,s_{i+1}) \leq \varepsilon \frac{E_{max}}{n}$ and $s_j \not\prec s_{j+1}$.*

Proof In the error-bounded setting GMS has to make as many merges as possible, yet introducing at most εE_{max} error. Let the total error introduced at some intermediate step be E_{tot}. GMS will elect s_i, s_{i+1} for merging at that step only if there is no other more similar pair among those in the intermediate result and $E_{tot} + \varepsilon \frac{E_{max}}{n} \leq \varepsilon E_{max}$. Since s_i, s_{i+1} are most similar in $\mathbf{s}_{j+1}$, at most $n-j-1$ more similar tuple pairs may exist in $\mathbf{s}_n \setminus \mathbf{s}_j$. Once all of these tuples are merged, the total error will be $E_{tot} \leq (n-j-1)\cdot \varepsilon \frac{E_{max}}{n}$. Since these merging steps do not change the similarity of s_i, s_{i+1} (due to $s_j \not\prec s_{j+1}$), s_i, s_{i+1} become now the most similar pair. Since $E_{tot} + \varepsilon \frac{E_{max}}{n} \leq \varepsilon E_{max}$, GMS will make at least one more merging step, choosing s_i, s_{i+1} for merging. □

To avoid the heap growing too much while we wait for a non-adjacent tuple pair, we adopt the same heuristic as for the size-bounded case. That is, the more tuples follow the current merge candidate, the more likely it is that this pair will be merged also by GMS. Hence, we use a user-specified parameter δ to determine the minimum number of tuples that should follow the merge candidate for it to be merged.

Figure 13 shows the *$gPTA_\varepsilon$* algorithm for the greedy evaluation of error-bounded PTA. The variable E_{tot} is updated after each merging step and tracks the total error made so far. In addition, in line 6 we estimate the maximal total error, $\overline{E}_{max}$, and the size of the ITA relation, $\overline{n}$. Incoming ITA tuples are inserted into the heap as they arrive and merging is attempted with each new tuple. Merging may only happen if the key of the node at the top of the heap is less than the average expected error, $\varepsilon\overline{E}_{max}/\overline{n}$, and the node is followed by a non-adjacent tuple pair or at least δ adjacent tuples. Once the whole ITA relation has been processed we know the real maximal error that can be made, E_{max}. Therefore, as long as the total error introduced so far, E_{tot}, does not exceed the error bound we use GMS to finalize the merging process. The worst-case time and space complexity of *$gPTA_\varepsilon$* is the same as of *$gPTA_c$*.

Example 22 We run *$gPTA_\varepsilon$* on the **proj** relation with $\varepsilon = 0.5$ and $\delta = 1$. We set $\overline{E}_{max} = E_{max}$ of the corresponding ITA result, which is 269 285.714, and $\overline{n} = n = 7$. Therefore, if we were to merge all the tuples, the average error we would make per step would be $\varepsilon E_{max}/n = 19\,234.69$. *$gPTA_\varepsilon$* reads the ITA tuples one by one and tries to merge those that introduce less than the expected average error. The first such

```
Algorithm: $gPTA_\varepsilon(\mathbf{r}, \mathbf{A}, \mathbf{F}, \delta, \varepsilon)$
$H \leftarrow$ new empty heap;
Initialize the ITA operator with $\mathbf{F}$, $\mathbf{A}$, and $\mathbf{r}$;
$LastGapId \leftarrow 0$; $BG \leftarrow 0$; $AG \leftarrow 0$;
$E_{tot} \leftarrow 0$; $E_{max} \leftarrow 0$;
Estimate $\overline{E}_{max}$ and $\overline{n}$;
while $s_i \leftarrow$ next tuple from $\mathscr{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r}$ do
    Lines 6 – 12 from $gPTA_c$ algorithm in Fig. 11;
    while true do
        $N \leftarrow$ PEEK();
        if $N.key > \varepsilon\overline{E}_{max}/\overline{n}$ then break;
        if $N.id < LastGapId$ then
            $BG \leftarrow BG - 1$;
            $E_{tot} \leftarrow E_{tot} + N.key$;
            MERGE();
        else if $N.id > LastGapId \wedge N$ has $\delta$ successors then
            $AG \leftarrow AG - 1$;
            $E_{tot} \leftarrow E_{tot} + N.key$;
            MERGE();
        else
            break;
while true do
    $N \leftarrow$ PEEK();
    if $N.key < \infty \wedge (E_{tot} + N.key)/E_{max} \leq \varepsilon$ then
        MERGE();
        $E_{tot} \leftarrow E_{tot} + N.key$;
    else
        break;
return $H$;
```

Fig. 13 Greedy algorithm, $gPTA_\varepsilon$, for error-bounded PTA.

candidates are s_2, s_3 (see Fig. 1(c)). However, the δ parameter dictates to read tuple s_4 before merging that pair.

Theorem 3 *The output of the gPTA$_\varepsilon$ algorithm with $\delta = \infty$ is identical to that of GMS if $\frac{\overline{E}_{max}}{\overline{n}} \leq \frac{E_{max}}{n}$.*

Proof The $gPTA_\varepsilon$ algorithm will consider merging a pair of tuples s_i, s_{i+1} only if $dsim(s_i, s_{i+1}) \leq \varepsilon\frac{\overline{E}_{max}}{\overline{n}} \leq \varepsilon\frac{E_{max}}{n}$. Therefore, as follows from Proposition 4, the algorithm will merge the same tuple pairs that GMS would. □

The estimated values $\overline{n}$ and $\overline{E}_{max}$ play an important role in the $gPTA_\varepsilon$ algorithm. First, the estimated values may influence the correctness of the final output. Second, the precision of the estimate affects the size of the heap. The estimation of the ITA result size is easy, since it can be at most twice as large as the argument relation, thus $\overline{n} = 2|\mathbf{r}| - 1$. Estimating the maximal error, $\overline{E}_{max}$, is more complicated. The key aspect to consider here is how precise should the estimate be. As long as $\overline{E}_{max} \leq E_{max}$, the estimate only influences the size of the heap. That is, when $\overline{E}_{max} \ll E_{max}$, none or very few early merges will take place. Thus, the heap will be filled with almost the entire ITA result before the merging will commence. On the other hand, when the error is overestimated, i.e., $E_{max} < \overline{E}_{max}$, we cannot guarantee that the result is the same as for GMS. It seems reasonable to sample the argument relation and compute the corresponding ITA result to obtain an error estimate. A more detailed investigation of this aspect is part of the future work.

7 Experimental Evaluation

In this section we experimentally evaluate the capability of PTA to reduce ITA results. We quantify the error that our solutions introduce and compare it to other related approaches. We investigate also the runtime performance and space requirements of our algorithms.

7.1 Setup and Data

We implemented the algorithms introduced in this paper as well as the ATC [2], APCA [7], DWT [24], and PAA [14] algorithms and Chebyshev polynomials [6] in Java^{TM} Version 6. The experiments run on a Linux machine with four AMD 2600MHz Opteron processors and 16GB of RAM. An Oracle 11g database running on the same machine is used as data storage medium.

For the experiments we used the following four data sets: the real-world Incumbents data set kindly donated by the University of Arizona, USA; the synthetic employee temporal data set (ETDS) donated by F. Wang [28]; a subset of real-world time series data from the UCR Time Series Data Repository [13]; and a synthetic dataset for large scale experiments. We evaluate PTA with various aggregation functions and grouping attributes and with a varying number of temporal gaps. Therefore, we issue different aggregation queries over the four base relations to get a total of 12 ITA relations with different attribute value distributions, number of aggregation groups, and temporal gaps (see Table 1).

The ETDS relation reports the evolution of employees in a company and contains 2 875 697 records. Each record stores employee number, sex, department, title, salary, and contract validity interval in months. The ITA queries over this relation are summarized in Tab. 1(a). Queries E1, E2, and E3 specify different aggregation functions over the salary attribute without any grouping, yielding ITA results of 6 394 tuples each. Since these relations have no temporal gaps nor aggregation groups, we have $c_{min} = 1$. In Query E4 we group by employee number and department. The corresponding ITA result contains more than 5 million tuples, which exceeds the size of the input relation.

The Incumbents relation records the change of employee salaries over time. It has 83 857 tuples, where each tuple records a project ID, department ID, salary, and time interval in months. The ITA queries I1, I2, and I3 in Tab. 1(b) group the base table by department and project and compute different aggregate functions over the salary attribute.

Table 1 ITA aggregation queries used for the evaluation.

(a) ETDS relation

Name	Grouping, **A**	Agg. Functions, **F**	ITA Size	c_{min}
E1	$\emptyset$	*avg*(Salary)	6 394	1
E2	$\emptyset$	*max*(Salary)	6 394	1
E3	$\emptyset$	*sum*(Salary)	6 394	1
E4	Emp.No., Dep.	*avg*(Salary)	5 419 493	339 067

(b) Incumbents relation

Name	Grouping, **A**	Agg. Functions, **F**	ITA Size	c_{min}
I1	Dep., Proj.	*avg*(Salary)	16 144	131
I2	Dep., Proj.	*max*(Salary)	16 144	131
I3	Dep., Proj.	*sum*(Salary)	16 144	131

(c) Time series data

Name	Relation	No. of Dimensions	ITA Size	c_{min}
T1	Chaotic.dat	1	1 800	1
T2	Tide.dat	1	8 746	1
T3	Wind.dat	12	6 574	216

(d) Synthetic

Name	Grouping	Dimensions	ITA Size	c_{min}
S1	–	10	10 000 000	1
S2	yes	10	10 000 000	50 000

The UCR Time Series Data Repository [13] offers a variety of real-world time series data from various sources. We use two one-dimensional datasets, chaotic.dat and tide.dat, and one dataset with 12 dimensions, wind.dat. Each record in time series data has one or more aggregate values (dimensions) and a timestamp value. We replace the timestamp by a validity interval of length one to obtain a sequential relation. Thus, we can omit the ITA aggregation and pass the data directly to the PTA merging step. Table 1(c) summarizes the used time series.

To avoid any data induced bias we generate a synthetic dataset with 10 million tuples, one grouping attribute, and 10 aggregate attributes with uniformly distributed values. We issue two different ITA queries over this dataset as shown in Table 1(d). Query S1 does not specify any grouping, and the result has no temporal gaps, hence $c_{min} = 1$. Query S2 uses grouping and produces a result relation with 50 000 groups with 200 tuples in each.

7.2 Quality Evaluation

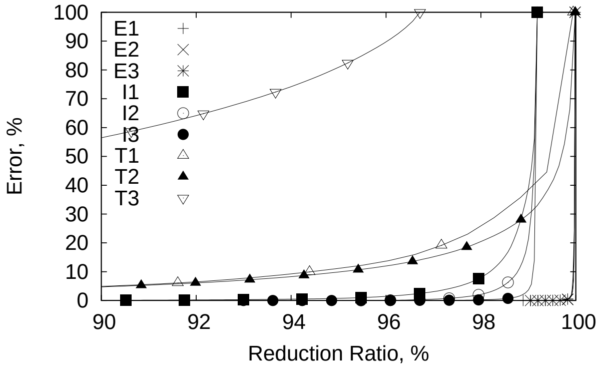
The first set of experiments evaluates the error that PTA introduces when reducing the ITA result using the DP and GMS based approaches.

7.2.1 Quality of the DP Approach

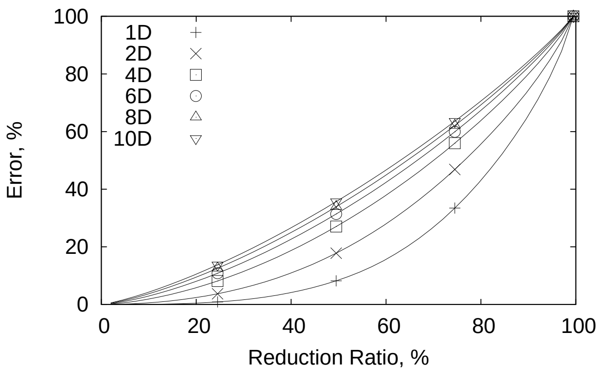
In Fig. 14 we measure the error introduced by the PTA_c algorithm for every possible output size. (Note that the PTA_ε results are identical and we do not report them here.) We use every dataset except E4 and the synthetic one as they are too large to be processed in reasonable time with the DP approach. The error and output size are normalized to the range between 0 to 100%. Thus, we show error growth curves, where the horizontal axis depicts the reduction ratio and the vertical axis the error. Any ITA result has 0% reduction and thus 0% error; 100% error is reached when the dataset is reduced to c_{min}.

Fig. 14(a) depicts the error curves in the range of 90% to 100% reduction for the first three datasets. Observe that for most queries the error remains very low even when they are significantly reduced. For example, the series T1 can be reduced by 95% (i.e., to 100 tuples) before the total error exceeded 10%. Only T3 reaches an error of 55% already at 90% of reduction.

The relatively large error on the high-dimensional dataset, T3, suggest that the reduction capabilities depend on the dimensionality of the data. Therefore, in Fig. 14(b) we evaluate several PTA queries over a 2 000 tuple subset of the synthetic dataset. For each query we specify a different number of aggregation attributes (dimensions). As the dimensionality increases, the introduced error grows.



(a) EDTS, Incumbents, and Time series data



(b) Synthetic data

Fig. 14 PTA error as a function of the reduction ratio.

From the results of this experiment we conclude that in most cases the PTA operator can reduce the ITA result size significantly inducing only a small error. Moreover, the reduction capabilities do not depend on the aggregation functions or grouping attributes in the query, but on the data dimensionality. This is not surprising as the dimensionality related problems have long been known [3].

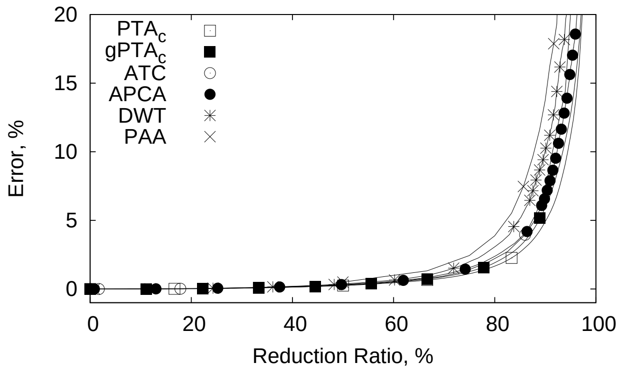
7.2.2 Quality of the Greedy Approach

We quantify the reduction error of the greedy PTA algorithms. First, we measure how close the greedy and the precise solutions are and whether the former can outperform other known data approximation algorithms. We use $gPTA_c$ with $\delta = \infty$ ($gPTA_\varepsilon$ yields identical results). Second, we evaluate the influence of δ on the results of the $gPTA_c$ and $gPTA_\varepsilon$ algorithms.

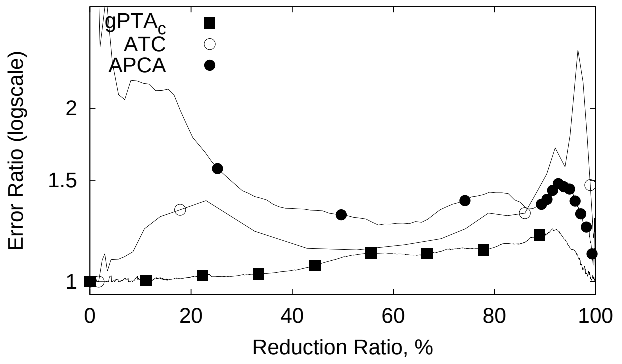
Figure 15 compares the errors introduced by $gPTA_c$ and other algorithms for Query T1 at each size bound from 1 to 1 800. In DWT there is no direct relationship between the number of coefficients retained and the number of segments in the restored time series signal. The signal restored from k coefficients will contain from k to $3k$ intervals. To obtain a DWT result of size c, we have to search for a suitable k. If several solutions exist, we retain the one that yields the smallest approximation error. APCA, on the other hand, will always yield c segments as it applies greedy merging on top of the time series that is reconstructed from wavelet coefficients. ATC takes as input an error bound. We generate a list of exponentially decaying error bounds and compute the ATC result for each one of them. If for two error bounds ATC yields several results of the same size, we keep the one that introduces the smallest error.

Observe in Figure 15(a) that the $gPTA_c$ error curve is closest to the error curve of the precise result which is computed with PTA_c. Thus, the greedy algorithm is closest to the optimal result and outperforms all other data approximation algorithms, out of which DWT and PAA perform significantly worse. We take the PTA_c result as a baseline and plot the ratio of the reduction error to the baseline. An error ratio 1 means an optimal reduction, whereas any greater value signifies a divergence from it. The error ratio of $gPTA_c$ is very close to 1 and increases only slightly with the number of merging steps to reach 1.25. This behavior is predicted by Theorem 1. ATC and APCA lag behind. DWT and PAA perform significantly worse and are not shown.

We run the same experiment for all queries and compute the average error ratio over the full range of c values for $gPTA_c$, ATC, APCA, DWT, and PAA algorithms. The result is shown in Fig. 16. A pattern-filled bar depicts the average error ratio, and a thin line indicates the standard error. Note the logarithmic scale for the error ratio. The figure reports also the average error ratio of the approximation with



(a) Error

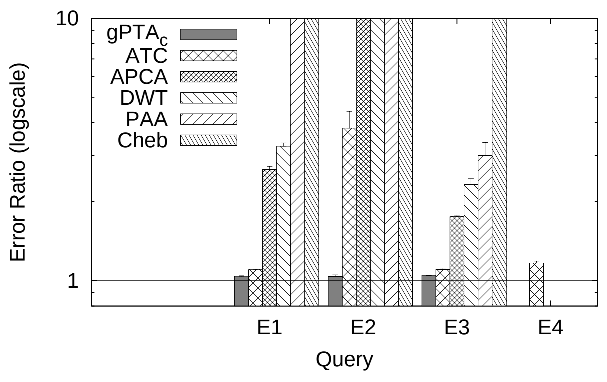


(b) Error ratio

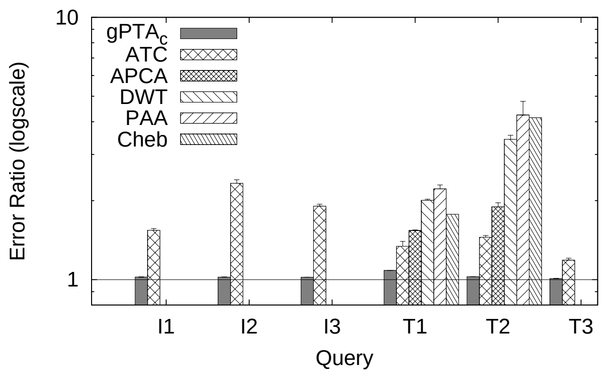
Fig. 15 Reduction error of different algorithms for query T1.

Chebyshev polynomials. We compute approximations using from 1 up to 1000 coefficients and compare the resulting time series with the PTA_c result with the same number of tuples. Overall, the $gPTA_c$ algorithm consistently provides the best error ratio, that is, its results are closest to those of PTA_c. ATC is the second best algorithm, however, its performance is not consistent; ATC shows satisfactory results for E4 and T3 but not for I1 and I2. For query E4 we use $gPTA_c$ as baseline since the dataset is too large to be evaluated with PTA_c, and we compare it with the error of ATC, which is slightly worse. APCA, DWT, PAA, and Chebyshev polynomials are not applicable for the queries I1, I2, I3, and T3 since they cannot cope with multiple aggregation groups and temporal gaps. Interestingly, the algorithms designed for time series approximation perform better on the time series data T1, T2, and T3 and significantly worse on temporal data, which can be explained by their inability to handle constant value intervals.

In our last experiment we evaluate the impact of δ to the quality of the result of $gPTA_c$ and $gPTA_\varepsilon$ algorithms. Instead of estimating the relation size and the total error we use the correct values. Figure 17 shows the average error ratio and standard error for a varying δ and different datasets. PTA_c and PTA_ε are used as baseline solution, and the er-

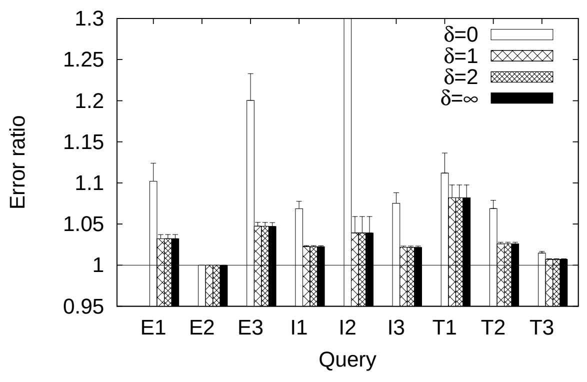


(a) ETDS data

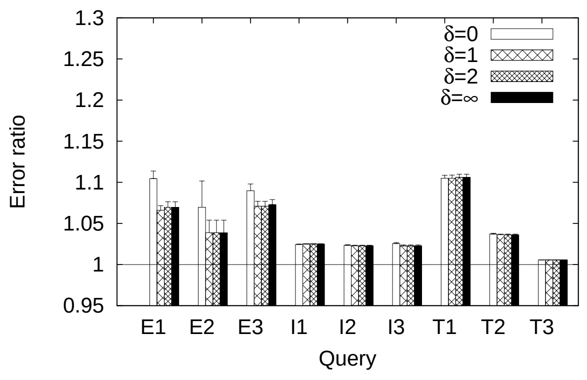


(b) Incumbents and Time Series data

Fig. 16 Average error ratio for different datasets.



(a) $gPTA_c$



(b) $gPTA_\varepsilon$

Fig. 17 Impact of δ.

ror is averaged over all possible size bounds, c, and error bounds, ε. When $\delta = 0$ the algorithms return the worst result, whereas for $\delta = \infty$ the best results that are possible with greedy merging are obtained. It is interesting to see that for $\delta \geq 1$ the results are practically the same. From this observation we can conclude that reading ahead by just one tuple (before merging) is sufficient to obtain very good results. In the following scalability experiments we show that a small δ allows also to reduce the heap significantly and to get better runtime performance.

7.3 Performance Evaluation

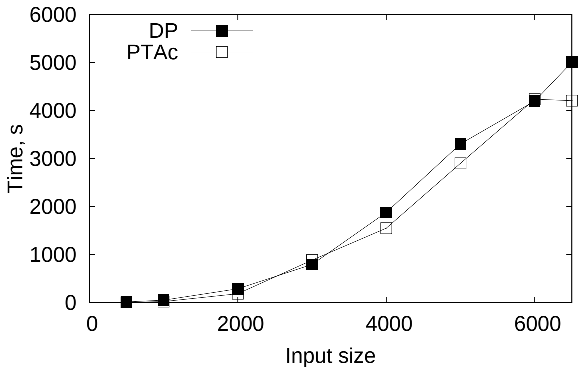
The second set of experiments evaluates the runtime performance. Thereby, we measure only the time of the merging phase and exclude the time taken to produce the ITA result and to write the final PTA result back to the database.

7.3.1 Performance of the DP Approach

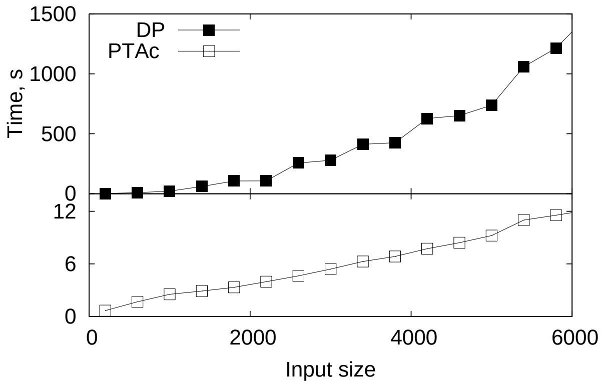
As a baseline solution (DP) we use the straightforward implementation of the DP scheme described in Sec. 5.1. We compare it to the PTA_c algorithm, which implements the improvements described in Sec. 5.2–5.3 (the same results are obtained for PTA_ε, and hence are not reported here). We expect PTA_c to be faster when dealing with data that sports multiple temporal gaps and aggregation groups.

Figure 18 illustrates the impact of the input size to the computation time. In Fig. 18(a), we use sequential subsets of the synthetic dataset, which have no temporal gaps and aggregation groups. The size of the input varies from 500 to 6500 tuples, whereas the output, c, and dimensionality, p, are fixed at 500 and 10, respectively. As expected, the two approaches show no significant difference. Figure 18(b) shows that the performance of PTA_c improves significantly over DP when the argument relation sports multiple groups or gaps. Here we use subsets of different size of the grouped synthetic data (S2) and we keep the number of aggregation groups fixed at 200; only the number of tuples within each group increases with the input size. As the figure shows, PTA_c significantly outperforms DP and scales almost linearly since the presence of gaps reduces the amount of computation.

The next experiments in Fig. 19 shows how the change of the output size, c, influences the performance. As input data we use 2 000 tuples from the synthetic dataset with 200 groups and 10 tuples in each group. The value of c varies from 1 to 2 000. As expected, the running time increases lin-



(a) Synthetic data with no gaps (S1)



(b) Synthetic data with gaps (S2)

Fig. 18 Runtime as a function of the input size.

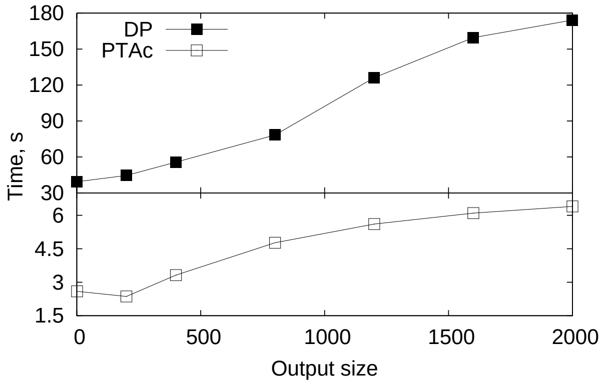


Fig. 19 Runtime as a function of the output size on synthetic data with gaps.

early with the increasing size of the output. Observe, however, that the PTA_c algorithm is not overly sensitive to the size bound, c, as the presence of gaps is the most important speed factor.

To summarize, the experiments confirm the estimated performance of the PTA_c algorithm. When data sports temporal gaps or aggregation groups are specified in the query (as in many realistic applications), PTA_c is much faster than the plain DP approach.

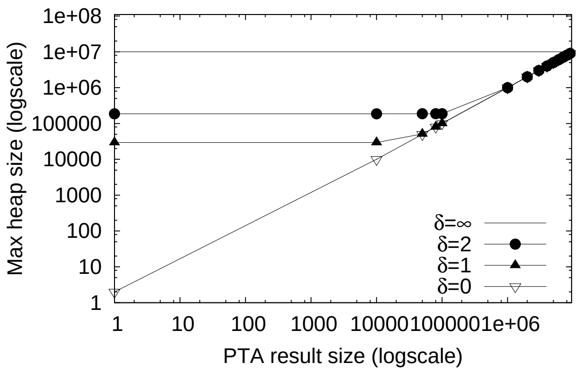
7.3.2 Performance of the Greedy Approach

First, we quantify the space requirements of the two greedy algorithms by measuring the maximal heap size. We use the synthetic relation without gaps and vary the output size as well as δ. The input size is fixed at 10 000 000. The results are shown in Fig. 20, where the horizontal axis ranges over the result size and the vertical axis over the heap size. In $gPTA_c$, the heap is filled with the whole ITA result when $\delta = \infty$. When $\delta = 0$, the behavior is exactly the opposite, and the heap size never exceeds the output size, c. For other values of δ the heap size is $c + \beta$. Smaller δ lead to smaller β. Eventually β converges to 0 and the heap size scales linearly with respect to the output size, c. The $gPTA_\varepsilon$ algorithm behaves similarly, however, its heap is significantly larger independently of δ.

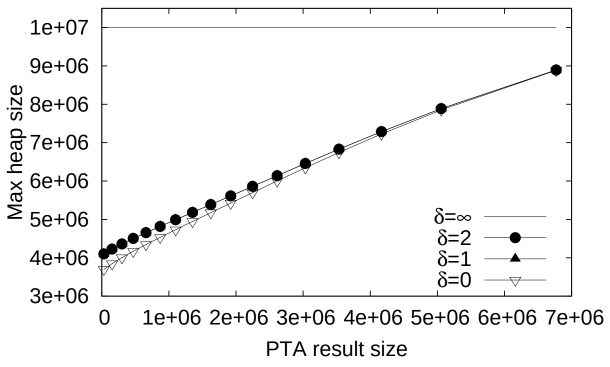
The last experiment in Fig. 21 compares the performance of $gPTA_\varepsilon$ and $gPTA_c$ to APCA, DWT, PAA, and ATC by measuring the average running time of each algorithm with respect to the size of the input. We use subsets of the synthetic dataset without temporal gaps that range in size from 1 to 10 million tuples. The size bound, c, for the $gPTA_c$ algorithm is set to 10% of the input size; we found empirically that the corresponding error bound for $gPTA_\varepsilon$ is $\varepsilon = 0.65$. We use $\delta = 1$ since previous experiments have shown that this value yields very good quality at the smallest space requirements. For ATC we set the local error bound 0.01. We exclude the runtime of approximation with Chebyshev polynomials. The algorithm needs $O(nc)$ time to compute c coefficients making it unsuitable for large datasets and large values of c. As can be seen in Fig. 21, $gPTA_\varepsilon$ is the slowest algorithm since it has to deal with an ever-increasing heap structure. On the other hand, $gPTA_c$ is comparable to other approaches. Such performance advantage is due to very small heap that $gPTA_c$ operates on.

7.4 Summary

The experimental results reported in this section show that the PTA operator can significantly reduce the ITA result, yet introducing only small errors. The dynamic programming based algorithms, PTA_c and PTA_ε, scale linearly in real-world situations, though in the worst case they remain quadratic with respect to the input size. The reduction of the ITA result obtained with the two greedy algorithms, $gPTA_c$ and $gPTA_\varepsilon$, is very close to the optimal result. The greedy algorithms consistently and significantly outperform other known approximation methods in terms of approximation quality. In addition, they are scalable for huge datasets.



(a) *gPTA$_c$*



(b) *gPTA$_\varepsilon$*

Fig. 20 Maximal heap size of $gPTA_c$ and $gPTA_\varepsilon$ as a function of the output size.

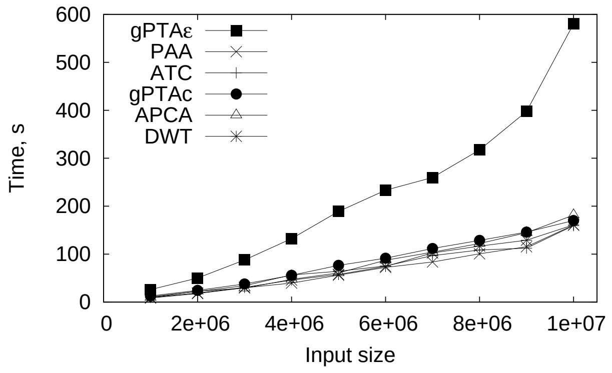


Fig. 21 Performance of the greedy algorithms compared to other linear approximation methods.

8 Conclusions and Future Work

In this paper we defined size- and error-bounded parsimonious temporal aggregation (PTA). This new aggregation operator overcomes limitations of existing operators and reduces the result size of instant temporal aggregation by merging the most similar tuples until a user-specified error or size bound is satisfied. We presented two dynamic programming based evaluation algorithms, PTA_c and PTA_ε, to compute a precise result for PTA queries. In realistic situations, when data sports temporal gaps or aggregation groups are specified in the query, the algorithms scale linearly with respect to the size of the input. For a quick computation of an approximation of the PTA result, we proposed two greedy algorithms, $gPTA_c$ and $gPTA_\varepsilon$. We proved that the error ratio of the greedy approach with respect to the precise solution is upper-bounded by $O(\log n)$, where n is the size of ITA result. The greedy algorithms take $O(c+\beta)$ space and $O(n\log(c+\beta))$ time for a result of size c, and β is typically very small. An extensive experimental evaluation confirmed the theoretical estimations and showed that the greedy algorithms scale very well for large data sets and provide significantly better approximation quality than other known approximation techniques.

In our future work we will explore the possibility of merging tuples separated by temporal gaps. In addition, we will address the issue of estimating the maximal error that the reduction of an ITA result may introduce. We believe that novel ways to sample temporal data have to be developed in order to obtain good estimates. We will extend both greedy algorithms to deal with streaming temporal data. This is a challenging problem since a streaming ITA result cannot be sorted along the aggregation groups. Finally, a careful investigation of different error measures is worthwhile.

Acknowledgements

We are grateful to the anonymous reviewers for their in-depth reviews of the article and valuable comments. This work was funded in part by the Swiss National Science Foundation (SNSF) through the Tameus project (proposal no 200021_135361).

References

1. Agrawal, R., Faloutsos, C., Swami, A.: Efficient search in sequence databases. In: Proc. of the 4th Int. Conf. on Foundations of Data Organization and Algorithms (1993)
2. Berberich, K., Bedathur, S.J., Neumann, T., Weikum, G.: A time machine for text search. In: Proc. of the 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 519–526 (2007)
3. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: Proc. of the 7th Int. Conf. on Database Theory, pp. 217–235 (1999)
4. Böhlen, M.H., Gamper, J., Jensen, C.S.: Multi-dimensional aggregation for temporal data. In: Proc. of the 10th Int. Conf. on Extending Database Technology, pp. 257–275. Springer (2006)
5. Böhlen, M.H., Snodgrass, R.T., Soo, M.D.: Coalescing in temporal databases. In: Proc. of the 22th Int. Conf. on Very Large Data Bases, pp. 180–191 (1996)
6. Cai, Y., Ng, R.: Indexing spatio-temporal trajectories with Chebyshev polynomials. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 599–610. ACM (2004)

7. Chakrabarti, K., Keogh, E., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. ACM Trans. Database Syst. **27**(2), 188–228 (2002)
8. Elmeleegy, H., Elmagarmid, A.K., Cecchet, E., Aref, W.G., Zwaenepoel, W.: Online piece-wise linear approximation of numerical streams with precision guarantees. PVLDB **2**(1), 145–156 (2009)
9. Gordevicius, J., Gamper, J., Böhlen, M.H.: A greedy approach towards parsimonious temporal aggregation. In: 15th International Symposium on Temporal Representation and Reasoning, TIME, pp. 88–92 (2008)
10. Gordevicius, J., Gamper, J., Böhlen, M.H.: Parsimonious temporal aggregation. In: EDBT, pp. 1006–1017 (2009)
11. Jagadish, H.V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K.C., Suel, T.: Optimal histograms with quality guarantees. In: VLDB'98, pp. 275–286 (1998)
12. Keogh, E., Kasetty, S.: On the need for time series data mining benchmarks: a survey and empirical demonstration. Data Mining and Knowledge Discovery **7**(4), 349–371 (2003)
13. Keogh, E., Xi, X., Wei, L., Ratanamahatana, C.: The UCR time series classification/clustering repository: www.cs.ucr.edu/∼ eamonn/time_series_data/. Accessed on april 15, 2009
14. Keogh, E.J., Pazzani, M.J.: A simple dimensionality reduction technique for fast similarity search in large time series databases. In: In 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD, pp. 122–133. Springer (2000)
15. Kline, N., Snodgrass, R.T.: Computing temporal aggregates. In: ICDE Proceedings, pp. pp. 222–231 (1995)
16. Li, C.S., Yu, P., Castelli, V.: Hierarchyscan: a hierarchical similarity search algorithm for databases of long sequences. In: Proceedings of the Twelfth International Conference on Data Engineering, pp. 546–553 (1996)
17. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing SAX: a novel symbolic representation of time series. Data Min. Knowl. Discov. **15**(2), 107–144 (2007)
18. Moon, B., Vega Lopez, I.F., Immanuel, V.: Efficient algorithms for large-scale temporal aggregation. IEEE Transactions on Knowledge and Data Engineering **15**(3), pp. 744–759 (2003)
19. Navathe, S.B., Ahmed, R.: A temporal relational model and a query language. Inf. Sci. **49**(1-3), 147–175 (1989)
20. Palpanas, T., Vlachos, M., Keogh, E., Gunopulos, D.: Streaming time series summarization using user-defined amnesic functions. IEEE Transactions on Knowledge and Data Engineering pp. 992–1006 (2008)
21. Palpanas, T., Vlachos, M., Keogh, E., Gunopulos, D., Truppel, W.: Online amnesic approximation of streaming time series. In: Data Engineering, 2004. Proceedings. 20th International Conference on, pp. 339–349. IEEE (2004)
22. Shieh, J., Keogh, E.: iSAX: indexing and mining terabyte sized time series. In: KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 623–631 (2008)
23. Snodgrass, R.T., Gomez, S., McKenzie, L.E.: Aggregates in the temporal query language TQuel. IEEE Trans. Knowl. Data Eng. **5**(5), 826–842 (1993)
24. Stollnitz, E., DeRose, A., Salesin, D.: Wavelets for computer graphics: a primer, part 1. Computer Graphics and Applications, IEEE **15**(3), 76–84 (1995)
25. Tao, Y., Papadias, D., Faloutsos, C.: Approximate temporal aggregation. In: ICDE, pp. 190–201 (2004)
26. Tuma, P.: Implementing historical aggregates in TempIS. Ph.D. thesis, Wayne State University, Detroit, Michigan (1992)
27. Vega Lopez, I.F., Snodgrass, R.T., Moon, B.: Spatiotemporal aggregate computation: A survey. IEEE Transactions on Knowledge and Data Engineering **17**(2), pp. 271–286 (2005)
28. Wang, F.: Employee temporal data set. http://timecenter.cs.aau.dk/
29. Wettschereck, D., Aha, D.W., Mohri, T.: A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artif. Intell. Rev. **11**(1-5), 273–314 (1997)
30. Yang, J., Widom, J.: Incremental computation and maintenance of temporal aggregates. The VLDB Journal (2003)
31. Yi, B.K., Faloutsos, C.: Fast time sequence indexing for arbitrary Lp norms. In: VLDB '00, pp. 385–394. Morgan Kaufmann Publishers Inc. (2000)