M4: A Visualization-Oriented Time Series Data Aggregation

Uwe Jugel, Zbigniew Jerzak,
Gregor Hackenbroich
SAP AG
Chemnitzer Str. 48, 01187 Dresden, Germany
{firstname}.{lastname}@sap.com

Volker Markl
Technische Universität Berlin
Straße des 17. Juni 135
10623 Berlin, Germany
volker.markl@tu-berlin.de

ABSTRACT

Visual analysis of high-volume time series data is ubiquitous in many industries, including finance, banking, and discrete manufacturing. Contemporary, RDBMS-based systems for visualization of high-volume time series data have difficulty to cope with the hard latency requirements and high ingestion rates of interactive visualizations. Existing solutions for lowering the volume of time series data disregard the semantics of visualizations and result in visualization errors.

In this work, we introduce M4, an aggregation-based time series dimensionality reduction technique that provides error-free visualizations at high data reduction rates. Focusing on line charts, as the predominant form of time series visualization, we explain in detail the drawbacks of existing data reduction techniques and how our approach outperforms state of the art, by respecting the process of line rasterization.

We describe how to incorporate aggregation-based dimensionality reduction at the query level in a visualization-driven query rewriting system. Our approach is generic and applicable to any visualization system that uses an RDBMS as data source. Using real world data sets from high tech manufacturing, stock markets, and sports analytics domains we demonstrate that our visualization-oriented data aggregation can reduce data volumes by up to two orders of magnitude, while preserving perfect visualizations.

Keywords: Relational databases, Query rewriting, Dimensionality reduction, Line rasterization

1. INTRODUCTION

Enterprises are gathering petabytes of data in public and private clouds, with *time series* data originating from various sources, including sensor networks [15], smart grids, financial markets, and many more. Large volumes of collected time series data are subsequently stored in relational databases. Relational databases, in turn, are used as backend by visual data analysis tools. Data analysts interact with the visualizations and their actions are transformed by the visual data analysis tools into a series of queries that are issued against the relational database, holding the original time series data. In state-of-the-art visual analytics tools, e.g., Tableau, QlikView, SAP Lumira, etc., such queries are issued to the database without considering the cardinality of the query result. However, when reading data from high-volume data sources, result sets often contain millions of rows. This leads to very high bandwidth consumption between the visualization system and the database.

Let us consider the following example. SAP customers in high tech manufacturing report that it is not uncommon for 100 engineers to simultaneously access a global database, containing equipment monitoring data. Such monitoring data originates from sensors embedded within the high tech manufacturing machines. The common reporting frequency for such embedded sensors is 100Hz [15]. An engineer usually accesses data which spans the last 12 hours for any given sensor. If the visualization system uses a non-aggregating query, such as

```
SELECT time,value FROM sensor WHERE time > NOW()-12*3600
```

to retrieve the necessary data from the database, the total amount of data to transfer is $100_{\text{users}} \cdot (12 \cdot 3600)_{\text{seconds}} \cdot 100_{Hz} = 432$ million rows, i.e., over 4 million rows per visualization client. Assuming a wire size of 60 bytes per row, the total amount of data that needs to be transferred from the database to all visualization clients is almost 26GB. Each user will have to wait for nearly 260MB to be loaded to the visualization client before he or she can examine a chart, showing the sensor signal.

With the proliferation of high frequency data sources and real-time visualization systems, the above concurrent-usage pattern and its implications are observed by SAP not only in high tech manufacturing, but across a constantly increasing number of industries, including sports analytics [22], finance, and utilities.

The final visualization, which is presented to an engineer, is inherently restricted to displaying the retrieved data using *width* $\times$ *height* pixels - the area of the resulting chart. This implies that a visualization system must perform a data reduction, transforming and projecting the received result set onto a *width* $\times$ *height* raster. This reduction is performed implicitly by the visualization client and is applied to all result sets, regardless of the number of rows they contain. The goal of this paper is to leverage this fundamental observation and apply an appropriate data reduction already at the query level within the database. As illustrated in Figure 1, the goal is to rewrite a visualization-related query Q using a data reduction operator M_R, such that the resulting query

Proceedings of the VLDB Endowment, Vol. 7, No. 10

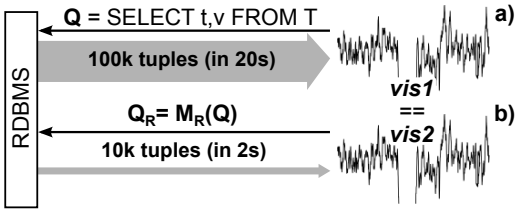


Figure 1: Time series visualization: a) based on a unbounded query without reduction; b) using visualization-oriented reduction at the query level.

Q_R produces a much smaller result set, without impairing the resulting visualization. Significantly reduced data volumes mitigate high network bandwidth requirements and lead to shorter waiting times for the users of the visualization system. Note that the goal of our approach is not to compute images inside the database, since this prevents client-side interaction with the data. Instead, our system should select subsets of the original result set that can be consumed transparently by any visualization client.

To achieve these goals, we present the following contributions. We first propose a visualization-driven query rewriting technique, relying on relational operators and parametrized with *width* and *height* of the desired visualization. Secondly, focusing on the detailed semantics of line charts, as the predominant form of time series visualization, we develop a visualization-driven aggregation that only selects data points that are necessary to draw the correct visualization of the complete underlying data. Thereby, we model the visualization process by selecting for every time interval, which corresponds to a pixel column in the final visualization, the tuples with the *minimum* and *maximum value*, and additionally the *first* and *last* tuples, having the *minimum* and *maximum timestamp* in that pixel column. To best of our knowledge, there is no application or previous discussion of this data reduction model in the literature, even though it provides superior results for the purpose of line visualizations. In this paper, we remedy this shortcoming and explain the importance of the chosen *min*, *max*, and the additional *first* and *last* tuples, in context of line rasterization. We prove that the pixel-column-wise selection of these four tuples is required to ensure an error-free two-color (binary) line visualization. Furthermore, we denote this model as *M4 aggregation* and discuss and evaluate it for line visualizations in general, including anti-aliased (non-binary) line visualizations.

Our approach significantly differs from the state-of-the-art time series dimensionality reduction techniques [11], which are often based on line simplification algorithms [25], such as the Ramer-Douglas-Peucker [6, 14] and the Visvalingam-Whyatt algorithms [26]. These algorithms are computationally expensive $\mathcal{O}(n \log(n))$ [19] and disregard the projection of the line to the *width* × *height* pixels of the final visualization. In contrast, our approach has the complexity of $\mathcal{O}(n)$ and provides perfect visualizations.

Relying only on relational operators for the data reduction, our visualization-driven query rewriting is generic and can be applied to any RDBMS system. We demonstrate the improvements of our techniques in a real world setting, using prototype implementations of our algorithms on top of SAP HANA [10] and Postgres (postgres.org).

The remainder of the paper is structured as follows. In Section 2, we present our system architecture and describe our query rewriting approach. In Section 3, we discuss our focus on line charts. Thereafter, in Section 4, we provide the details of our visualization-oriented data aggregation model and discuss the proposed *M4 aggregation*. After describing the drawbacks of existing time series dimensionality reduction techniques in Section 5, we compare our approach with these techniques and evaluate the improvements regarding query execution time, data efficiency, and visualization quality in Section 6. In Section 7, we discuss additional related work, and we eventually conclude with Section 8.

2. QUERY REWRITING

In this section, we describe our query rewriting approach to facilitate data reduction for visualization systems that rely on relational data sources.

To incorporate operators for data reduction, an original query to a high-volume time series data source needs to be rewritten. The rewriting can either be done directly by the visualization client or by an additional query interface to the relational database management system (RDBMS). Independent of where the query is rewritten, the actual data reduction will always be computed by the database itself. The following Figure 2 illustrates this query rewriting and data-centric dimensionality reduction approach.

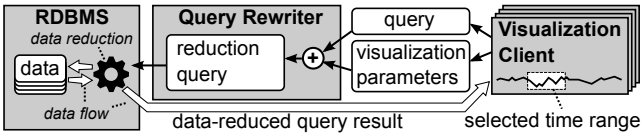


Figure 2: Visualization system with query rewriter.

Query Definition. The definition of a query starts at the visualization client, where the user first selects a time series data source, a time range, and the type of visualization. Data source and time range usually define the main parts of the original *query*. Most queries, issued by our visualization clients, are of the form `SELECT time, value FROM series WHERE time > t1 AND time < t2`. But practically, the visualization client can define an arbitrary relational query, as long as the result is a valid time series relation.

Time Series Data Model. We regard time series as binary relations $T(t, v)$ with two numeric attributes: timestamp $t \in \mathbb{R}$ and value $v \in \mathbb{R}$. Any other relation that has at least two numerical attributes can be easily projected to this model. For example, given a relation $X(a, b, c)$, and knowing that a is a numerical timestamp and b and c are also numerical values, we can derive two separate time series relations by means of projection and renaming, i.e., $T_b(t, v) = \pi_{t \leftarrow a, v \leftarrow b}(X)$ and $T_c(t, v) = \pi_{t \leftarrow a, v \leftarrow c}(X)$.

Visualization Parameters. In addition to the *query*, the visualization client must also provide the *visualization parameters* width w and height h, i.e., the exact pixel resolution of the desired visualization. Determining the exact pixel resolution is very important, as we will later show in Section 6. For most visualizations the user-selected chart size ($w_{chart} \times h_{chart}$) is different from the actual resolution ($w \times h$) of the canvas that is used for rasterization of the geometry. Figure 3 depicts this difference using a schematic example of a line chart that occupies 14×11 screen pixels in

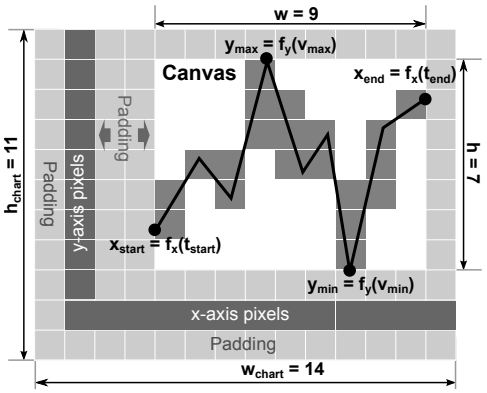
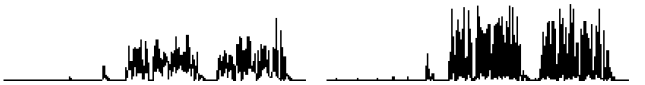


Figure 3: Determining the visualization parameters.

a) Conditional query to apply PAA data reduction

```
WITH Q  AS (SELECT t,v FROM sensors WHERE          1) original query Q
           id = 1 AND t >= $t1 AND t <= $t2),
     QC AS (SELECT count(*) c FROM Q)              2) cardinality query Q_C
SELECT * FROM Q WHERE (SELECT c FROM QC) <= 800    3a) use Q if low card.
UNION
SELECT * FROM (                                    reduction query Q_D:
     SELECT min(t),avg(v) FROM Q                   compute aggregates
     GROUP BY round(200*(t-$t1)/($t2-$t1))         for each pixel-column
) AS QD          WHERE (SELECT c FROM QC) >  800   3b) use Q_D if high card.
```



b) resulting image c) expected image

Figure 4: Query rewriting result and visualization.

total, but only uses 9×7 pixels for drawing the actual lines. When deriving the data reduction operators from these visualization parameters w and h, our query rewriter assumes the following.

Given the width w and height h of the canvas area, the visualization client uses the following geometric transformation functions $x = f_x(t)$ and $y = f_y(v)$, $x, y \in \mathbb{R}$ to project each timestamp t and value v to the visualization's coordinate system.

$$f_x(t) = w \cdot (t - t_{start})/(t_{end} - t_{start})$$
$$f_y(v) = h \cdot (v - v_{min})/(v_{max} - v_{min}) \quad (1)$$

The projected real-valued time series data is then traversed by the drawing routines of the visualization client to derive the discrete pixels. We assume the projected minimum and maximum timestamps and values of the selected time series (t_{start}, t_{end}, v_{min}, v_{max}) match exactly with the real-valued left, right, bottom, and top boundaries (x_{start}, x_{end}, y_{min}, y_{max}) of the canvas area, i.e., we assume that the drawing routines do not apply an additional vertical or horizontal translation or rescaling operation to data. In our evaluation, in Section 6, we will discuss potential issues of these assumptions.

Query Rewriting. In our system, the *query rewriter* handles all visualization-related queries. Therefore, it receives a query Q and the additional visualization parameters width w and height h. The goal of the rewriting is to apply an additional data reduction to those queries, whose result set size exceeds a certain limit. The result of the rewriting process is exemplified in Figure 4a. In general, such a *rewritten query* Q_R contains the following subqueries.

1) The original query Q,
2) a cardinality query Q_C on Q,
3) a cardinality check (conditional execution),
3a) to either use the result of Q directly, or
3b) to execute an additional data reduction Q_D on Q.

Our system composes all relevant subqueries into one single SQL query to ensure a fast query execution. Thereby, we firstly leverage that databases are able to reuse results of subqueries, and secondly assume that the execution time of counting the number of rows selected by the original query is negligibly small, compared to the actual query execution time. The two properties are true for most modern databases and the first facilitates the second. In practice, subquery reuse can be achieved via common table expression [23], as defined by the SQL 1999 standard.

The SQL example in Figure 4a uses a width of $w = 200$ pixels, a cardinality limit of $4 \cdot w = 800$ tuples, and it applies a data reduction using a simple piece-wise aggregate approximation (PAA) [18] – a naive measure for time series dimensionality reduction. The corresponding visualization is a line chart, for which we only consider the width w to define the PAA parameters. For such aggregation-based data reduction operators, we align the time intervals with the pixel columns to model the process of visualization at the query level. We use the same geometric transformation $x = f_x(t)$, as is used by the visualization client and *round* the resulting value to a discrete group key between 0 and $w = 200$, i.e., we use the following grouping function.

$$f_g(t) = round(w \cdot (t - t_{start})/(t_{end} - t_{start})) \quad (2)$$

Figure 4b shows the resulting image that the visualization client derived from the reduced data, and Figure 4c shows the resulting image derived from the raw time series data. The averaging aggregation function significantly changes the actual shape of the time series. In Section 4, we will discuss the utility of different types of aggregation functions for the purpose of visualization-oriented data reduction. For now, let us focus on the overall structure of the query. Note that we model the described conditional execution using a union of the different subqueries (3a) and (3b) that have contradictory predicates based on the cardinality limit. This allows us to execute any user-defined query logic of the query Q only once. The execution of the query Q and the data reduction Q_D is all covered by this single query, such that no high-volume data needs to be copied to an additional data reduction or compression component.

3. TIME SERIES VISUALIZATION

In this work, we focus on *line charts* of high-volume time series data. The following short digression on time series visualization will explain our motivation.

There exist dozens of ways to visualize time series data, but only a few of them work well with large data volumes. Bar charts, pie charts, and similar simple chart types consume too much space per displayed element [8]. The most common charts that suffice for high-volume data are shown in Figure 5. These are *line charts* and *scatter plots* where a

single data point can be presented using only a few pixels. Regarding space efficiency, these two simple chart types are only surpassed by space-filling approaches [17]. However, the most ubiquitous visualization is the line chart; found in spreadsheet applications, data analytics tools, charting frameworks, system monitoring applications, and many more.

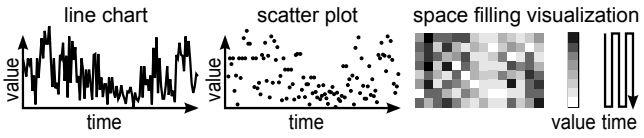


Figure 5: Common time series visualizations.

To achieve an efficient data reduction, we need to consider how a visualization is created from the underlying data. For *scatter plots* this process requires to *shift*, *scale* and *round* the time series relation $T(t, v)$ with $t, v \in \mathbb{R}$ to a relation of discrete pixels $V(x, y)$ with $x \in [1, w]$, $y \in [1, h]$. As already described in Section 2, we can reuse this geometric transformation for data reduction. Space-filling visualizations are similar. They also project a time series $T(t, v)$ to a sequence of discrete values $V(i, l)$ with $i \in [1, w \cdot h]$, $l \in [0, 255]$, where i is the position in the sequence and l is the luminance to be used.

An appropriate data reduction for scatter plots is a two-dimensional grouping of the time series, having a one-to-one mapping of pixels to groups. As a result, scatter plots (and also space-filling approaches) require selecting up to $w \cdot h$ tuples from the original data to produce correct visualizations. This may add up to several hundred thousand tuples, especially when considering today's high screen resolutions. The corresponding data reduction potential is limited.

This is not the case for line charts of high-volume time series data. In Section 4.4, we will show that there is an *upper bound* of $4 \cdot w$ required tuples for two-color line charts.

4. DATA REDUCTION OPERATORS

The goal of our query rewriting system is to apply visualization-oriented data reduction in the database by means of relational operators, i.e., using data aggregation. In the literature, we did not find any comprehensive discussion that describes the effects of data aggregation on rasterized line visualizations. Most time series dimensionality reduction techniques [11] are too generic and are not designed specifically for line visualizations. We will now remedy this shortcoming and describe several classes of operators that we considered and evaluated for data reduction and discuss their utility for line visualizations.

4.1 Visualization-Oriented Data Aggregation

As already described in Section 2, we model data reduction using a time-based grouping, aligning the time intervals with the pixel columns of the visualization. For each interval and thus for each pixel column we can compute aggregated values using one of the following options.

Normal Aggregation. A simple form of data reduction is to compute an aggregated value and an aggregated timestamp using the aggregation functions *min*, *max*, *avg*, *median*, *medoid*, or *mode*. The resulting data reduction queries on a time series relation $T(t, v)$, using a (horizontal) grouping function f_g and two aggregation functions f_t and f_v can be defined in relational algebra:

$$_{f_g(t)}G_{f_t(t),f_v(v)}(T) \tag{3}$$

We already used this simple form of aggregation in our query rewriting example (Figure 4), selecting a minimum (first) timestamp and an average value to model a piece-wise aggregate approximation (PAA) [18]. But any averaging function, i.e., using *avg*, *median*, *medoid*, or *mode*, will significantly distort the actual shape of the time series (see Figure 4b vs. 4c). To preserve the shape of the time series, we need to focus on the extrema of each group. For example, we want to select those tuples that have the minimum value v_{min} or maximum value v_{max} per group. Unfortunately, the grouping semantics of the relational algebra does not allow selection of non-aggregated values.

Value Preserving Aggregation. To select the corresponding tuples, based on the computed aggregated values, we need to join the aggregated data again with the underlying time series. Therefore, we replace one of the aggregation functions with the time-based group key (result of f_g) and join the aggregation results with T on that group key and on the aggregated value or timestamp. In particular, the following query

$$\pi_{t,v}(T \bowtie_{f_g(t)=k \wedge v=v_g} (_{f_{g(t)}}G_{k \leftarrow f_g(t), v_g \leftarrow f_v(v)}(T))) \tag{4}$$

selects the corresponding timestamps t for each aggregated value $v_g = f_v(v)$, and the following query

$$\pi_{t,v}(T \bowtie_{f_g(t)=k \wedge t=t_g} (_{f_{g(t)}}G_{k \leftarrow f_g(t), t_g \leftarrow f_t(t)}(T))) \tag{5}$$

selects the corresponding values v for each aggregated timestamp $t_g = f_t(t)$. Note that these queries may select more than one tuple per group, if there are duplicate values or timestamps per group. However, in most of our high-volume time series data sources, timestamps are unique and the values are real-valued numbers with multiple decimals places, such that the average number of duplicates is less than one percent of the overall data. In scenarios with more significant duplicate ratios, the described queries (4) and (5) need to be encased with additional compensating aggregation operators to ensure appropriate data reduction rates.

Sampling. Using the value preserving aggregation we can express simple forms of systematic sampling, e.g., selecting every first tuple per group using the following query.

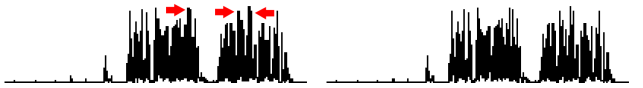
$$\pi_{t,v}(T \bowtie_{f_g(t)=k \wedge t=t_{min}} (_{f_{g(t)}}G_{k \leftarrow f_g(t), t_{min} \leftarrow min(t)}(T)))$$

For query level random sampling, we can also combine the value preserving aggregation with the SQL 2003 concept of the TABLESAMPLE or a selection operator involving a *random*() function. While this allows us to conduct data sampling inside the database, we will show in our evaluation that these simple forms sampling are not appropriate for line visualizations.

Composite Aggregations. In addition to the described queries (3), (4), and (5) that yield a single aggregated value per group, we also consider composite queries with several aggregated values per group. In relational algebra, we can model such queries as a union of two or more aggregating sub queries that use the same grouping function. Alternatively, we can modify the queries (4) and (5) to select multiple aggregated values or timestamps per group before joining again with the base relation. We then need to combine the

a) value-preserving MinMax aggregation query

```
SELECT t,v FROM Q JOIN
(SELECT round($w*(t-$t1)/($t2-$t1)) as k, -- define key
        min(v) as v_min, max(v) as v_max  -- get min,max
        FROM Q GROUP BY k) as QA          -- group by k
ON k = round($w*(t-$t1)/($t2-$t1))        -- join on k
   AND (v = v_min OR v = v_max)           -- &(min|max)
```

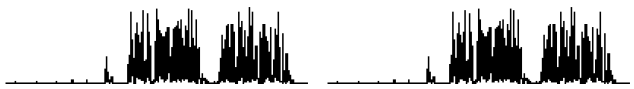


b) resulting image c) expected image

Figure 6: MinMax query and resulting visualization.

a) value-preserving M4 aggregation query

```
SELECT t,v FROM Q JOIN
(SELECT round($w*(t-$t1)/($t2-$t1)) as k, --define key
        min(v) as v_min, max(v) as v_max, --get min,max
        min(t) as t_min, max(t) as t_max  --get 1st,last
        FROM Q GROUP BY k) as QA          --group by k
ON k = round($w*(t-$t1)/($t2-$t1))        --join on k
   AND (v = v_min OR v = v_max OR         --&(min|max|
        t = t_min OR t = t_max)           --  1st|last)
```



b) resulting image == expected image

Figure 7: M4 query and resulting visualization.

join predicates, such that all different aggregated values or timestamps are correlated to either their missing timestamp or their missing value. The following MinMax aggregation will provide an example of a composite aggregation.

MinMax Aggregation. To preserve the shape of a time series, the first intuition is to group-wise select the vertical extrema, which we denote as *min* and *max tuples*. This is a composite value-preserving aggregation, exemplified by the SQL query in Figure 6a. The query projects existing timestamps and values from a time series relation Q, after joining it with the aggregation results QA. Q is defined by an original query, as already shown in Figure 4a. The group keys k are based on the rounded results of the (horizontal) geometric transformation of Q to the visualization's coordinate system. As stated before in Section 2, it is important that this geometric transformation is exactly the same transformation as conducted by visualization client, before passing the rescaled time series data to the line drawing routines for rasterization. Finally, the join of QA with Q is based on the group key k and on matching the values v in Q either with v_{min} or v_{max} from QA.

Figure 6b shows the resulting visualization and we now observe that it closely matches the expected visualization of the raw data 6c. In Section 4.3, we later discuss the remaining pixel errors; indicated by arrows in Figure 6b.

4.2 The M4 Aggregation

The composite *MinMax* aggregation focuses on the vertical extrema of each pixel column, i.e., of each corresponding time span. There are already existing approaches for selecting extrema for the purpose of data reduction and data analysis [12]. But most of them only partially consider the implications for data visualization and neglect the final projection of the data to discrete screen pixels.

A line chart that is based on a reduced data set, will always *omit lines* that would have connected the not selected tuples, and it will always *add new* approximating *lines* to bridge the not selected tuples between two consecutive selected tuples. The resulting errors (in the real-valued visualization space) are significantly reduced by the final discretization process of the drawing routines. This effect is the underlying principle of the proposed visualization-driven data reduction.

Intuitively, one may expect that selecting the minimum and maximum values, i.e., the tuples (t_{bottom}, $min(v)$) and (t_{top}, $max(v)$), from exactly w groups, is sufficient to derive a correct line visualization. This intuition is elusive, and this form of data reduction – provided by the MinMax aggregation – does not guarantee an error-free line visualization of the time series. It ignores the important *first* and *last* tuples of the each group. We now introduce the M4 aggregation that additionally selects these *first* and *last* tuples ($min(t)$, v_{first}) and ($max(t)$, v_{last}). In Section 4.3, we then discuss how M4 surpasses the MinMax intuition.

M4 Aggregation. M4 is a composite value-preserving aggregation (see Section 4.1) that groups a time series relation into w equidistant time spans, such that each group exactly corresponds to a pixel column in the visualization. For each group, M4 then computes the aggregates $min(v)$, $max(v)$, $min(t)$, and $max(t)$ – hence the name M4 – and then joins the aggregated data with the original time series, to add the missing timestamps t_{bottom} and t_{top} and the missing values v_{first} and v_{last}.

In Figure 7a, we present an example query using the M4 aggregation. This SQL query is very similar to the MinMax query in Figure 6a, adding only the $min(t)$ and $max(t)$ aggregations and the additional join predicates based on the *first* and *last* timestamps t_{min} and t_{max}. Figure 7b depicts the resulting visualization, which is now equal to the visualization of the unreduced underlying time series.

Complexity of M4. The required grouping and the computation of the aggregated values can be computed in $\mathcal{O}(n)$ for the n tuples of the base relation Q. The subsequent equi-join of the aggregated values with Q requires to match the n tuples in Q with $4 \cdot w$ aggregated tuples, using a hash-join in $\mathcal{O}(n+4 \cdot w)$, but w does not depend on n and is inherently limited by physical display resolutions, e.g., $w = 5120$ pixels for latest WHXGA displays. Therefore, the described M4 aggregation of has complexity of $\mathcal{O}(n)$.

4.3 Aggregation-Related Pixel Errors

In Figure 8 we compare three line visualizations: a) the schematic line visualization a time series Q, b) the visualization of $MinMax(Q)$, and c) the visualization of $M4(Q)$. $MinMax(Q)$ does not select the *first* and *last* tuples per pixel column, causing several types of line drawing errors.

In Figure 8b, the pixel (3,3) is not set correctly, since neither the start nor the end tuple of the corresponding line are included in the reduced data set. This kind of *missing line error* E_1 is distinctly visible with time series that have a very heterogeneous time distribution, i.e., notable gaps, resulting in pixel columns not holding any data (see pixel column 3 in Figure 8b). A missing line error is often exacerbated by

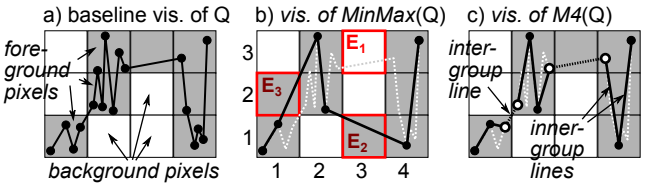


Figure 8: MinMax-related visualization error.

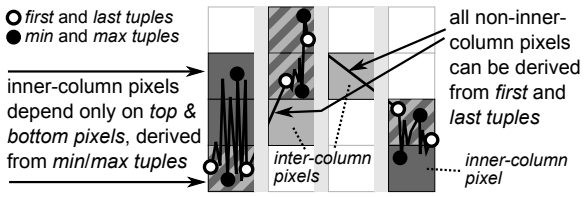


Figure 9: Illustration of the *Theorem 1*.

an additional *false line error* E_2, as the line drawing still requires to connect two tuples to bridge the empty pixel column. Furthermore, both error types may also occur in neighboring pixel columns, because the *inner-column lines* do not always represent the complete set of pixels of a column. Additional *inter-column pixels* – below or above the *inner-column pixels* – can be derived from the *inter-column lines*. This will again cause missing or additional pixels if the reduced data set does not contain the correct tuples for the correct inter-column lines. In Figure 8b, the MinMax aggregation causes such an error E_3 by setting the undesired pixel (1,2), derived from the *false line* between the maximum tuple in the first pixel column and the (consecutive) maximum tuple in the second pixel column. Note that these errors are independent of the resolution of the desired raster image, i.e., of the chosen number of groups. If we do not explicitly select the *first* and *last* tuples, we cannot guarantee that all tuples for drawing the correct inter-column lines are included.

4.4 The M4 Upper Bound

Based on the observation of the described errors, the question arises if selecting only the four extremum tuples for each pixel column guarantees an error-free visualization. In the following, we prove the existence of an upper bound of tuples necessary for an error-free, two-color line visualization.

Definition 1. A **width-based grouping** of an arbitrary time series relation $T(t, v)$ into w equidistant groups, denoted as $G(T) = (B_1, B_2, ..., B_w)$, is derived from T using the surjective grouping function $i = f_g(t) = round(w \cdot (t - t_{start})/(t_{end} - t_{start}))$ to assign any tuple of T to the groups B_i. A tuple (t, v) is assigned to B_i if $f_g(t) = i$.

Definition 2. A **width-based M4 aggregation** $G_{M4}(T)$ selects the *extremum* tuples $(t_{bottom_i}, v_{min_i})$, (t_{top_i}, v_{max_i}), (t_{min_i}, v_{first_i}), and (t_{max}, v_{last_i}) from each B_i of $G(T)$.

Definition 3. A **visualization relation** $V(x, y)$ with the attributes $x \in \mathbb{N}^{[1,w]}$ and $y \in \mathbb{N}^{[1,h]}$ contains all *foreground pixels* (x, y), representing all (black) pixels of all rasterized lines. $V(x, y)$ contains none of the remaining (white) *background pixels* in $\mathbb{N}^{[1,w]} \times \mathbb{N}^{[1,h]}$.

Definition 4. A **line visualization operator** $vis_{wh}(T) \rightarrow V$ defines, for all tuples (t, v) in T, the corresponding *foreground pixels* (x, y) in V.

Thereby vis_{wh} first uses the linear transformation functions f_x and f_y to rescale all tuples in T to the coordinate system $\mathbb{R}^{[1,w]} \times \mathbb{R}^{[1,h]}$ (see Section 2) and then tests for all discrete pixels and all non-discrete lines – defined by the consecutive tuples of the transformed time series – if a pixel is *on* the line or *not on* the line. For brevity, we omit a detailed description of line rasterization [2, 4], and assume the following two properties to be true.

Lemma 1. A rasterized line has no gaps.

Lemma 2. When rasterizing an *inner-column line*, no *foreground pixels* are set outside of the pixel column.

Theorem 1. Any *two-color line visualization* of an arbitrary time series T is equal to the *two-color line visualization* of a time series T' that contains at least the *four extrema* of all groups of the *width-based grouping* of T, i.e., $vis_{wh}(G_{M4}(T)) = vis_{wh}(T)$.

Figure 9 illustrates the reasoning of *Theorem 1*.

Proof. Suppose a visualization relation $V = vis_{wh}(T)$ represents the pixels of a *two-color line visualization* of an arbitrary time series T. Suppose a tuple can only be in one of the groups B_1 to B_w that are defined by $G(T)$ and that corresponds to the pixel columns. Then there is only one pair of consecutive tuples $p_j \in B_i$ and $p_{j+1} \in B_{i+1}$, i.e., only one *inter-column line* between each pair of consecutive groups B_i and B_{i+1}. But then, all other lines defined by the remaining pairs of consecutive tuples in T must define *inner-column lines*.

As of Lemmas 1 and 2, all *inner-column pixels* can be defined from knowing the *top* and *bottom* inner-column pixels of each column. Furthermore, since f_x and f_y are linear transformations, we can derive these *top* and *bottom* pixels of each column from the *min* and *max* tuples $(t_{bottom_i}, v_{min_i})$, (t_{top_i}, v_{max_i}) for each group B_i. The remaining *inter-column pixels* can be derived from all *inter-column lines*. Since there is only one *inter-column line* $\overline{p_j p_{j+1}}$ between each pair of consecutive groups B_i and B_{i+1}, the tuple $p_j \in B_i$ is the *last* tuple $(t_{max_i} v_{last_i})$ of B_i and $p_{j+1} \in B_{i+1}$ is the *first* tuple $(t_{min_{i+1}}, v_{first_{i+1}})$ of B_{i+1}.

Consequently, all pixels of the visualization relation V can be derived from the tuples $(t_{bottom_i}, v_{min_i})$, (t_{top_i}, v_{max_i}), $(t_{max_i} v_{last_i})$, (t_{min_i}, v_{first_i}) of each group B_i, i.e., $V = vis_{wh}(G_{M4}(T)) = vis_{wh}(T)$. □

Using *Theorem 1*, we can moreover derive

Theorem 2. There exists an *error-free two-color line visualization* of an arbitrary time series T, based on a subset T' of T, with $|T'| \leq 4 \cdot w$.

No matter how big T is, selecting the correct $4 \cdot w$ tuples from T allows us to create a perfect visualization of T. Clearly, for the purpose of line visualization, M4 queries provide data-efficient, predictable data reduction.

5. TIME SERIES DATA REDUCTION

In Section 4, we discussed averaging, systematic sampling and random sampling as measures for data reduction. They provide some utility for time series dimensionality reduction

in general, and may also provide useful data reduction for certain types of visualizations. However, for rasterized line visualizations, we have now proven the necessity of selecting the correct *min*, *max*, *first*, and *last* tuples. As a result, any averaging, systematic, or random approach will fail to obtain good results for line visualizations, if it does not ensure that these important tuples are included.

Compared to our approach, the most competitive approaches, as found in the literature [11], are times series dimensionality reduction techniques based on line simplification. For example, Fu et al. reduce a time series based on perceptually important points (PIP) [12]. Such approaches usually apply a distance measure defined between each three consecutive points of a line, and try to minimize this measure ϵ for a selected data reduction rate. This $min - \epsilon$ problem is complemented with a *min-#* problem, where the minimum number of points needs to be found for a defined distance ϵ. However, due to the $\mathcal{O}(n^2)$ worst case complexity [19] of this optimization problem, there are many heuristic algorithms for line simplification. The fastest class of algorithms works sequentially in $\mathcal{O}(n)$, processing every point of the line only once. The two other main classes have complexity of $\mathcal{O}(n \log(n))$ and either merge points until some error criterion is met (bottom up) or split the line (top down), starting with an initial approximating line, defined by the first and last points of the original line. The latter approaches usually provide a much better approximation of the original line [25]. Our aggregation-based data reduction, including M4, has a complexity of $\mathcal{O}(n)$.

Approximation Quality. To determine the approximation quality of a derived time series, most time series dimensionality reduction techniques use a geometric distance measure, defined between the segments of the approximating line and the (removed) points of the original line.

The most common measure is the Euclidean (perpendicular) distance, as shown in Figure 10a), which is the shortest distance of a point p_k to a line segment $\overline{p_i p_j}$. Other commonly applied distance measures are the area of the triangle (p_i, p_k, p_j) (Figure 10b) or the vertical distance of p_k to $\overline{p_i p_j}$ (Figure 10c).

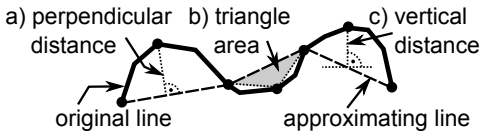


Figure 10: Distance measures for line simplification.

However, when visualizing a line using discrete pixels, the main shortcoming of the described measures is their generality. They are geometric measures, defined in $\mathbb{R} \times \mathbb{R}$, and do not consider the discontinuities in discrete 2D space, as defined by the cutting lines of two neighboring pixel rows or two neighboring pixel columns. For our approach, we consult the actual visualization to determine the approximation quality of the data reduction operators.

Visualization Quality. Two images of the same size can be easily compared pixel by pixel. A simple, commonly applied error measure is the mean square error MSE $= \frac{1}{wh} \sum_{x=1}^{w} \sum_{y=1}^{h} (I_{x,y}(V_1) - I_{x,y}(V_2))^2$, with $I_{x,y}$ defining the luminance value of a pixel (x, y) of a visualization V. Nonetheless, Wang et al. have shown [27] that MSE-based measures, including the commonly applied peak-signal-to-noise-ratio (PSNR) [5], do not approximate the model of human perception very well and developed the Structural Similarity Index (SSIM). For brevity and due to the complexity of this measure, we have to pass on without a detailed description of this measure. The SSIM yields a similarity value between 1 and -1. The related normalized distance measure between two visualizations V_1 and V_2 is defined as:

$$DSSIM(V_1, V_2) = \frac{1 - SSIM(V_1, V_2)}{2} \tag{6}$$

We use DSSIM to evaluate the quality of a line visualization that is based on a reduced time series data set, comparing it with the original line visualization of the underlying unreduced time series data set.

6. EVALUATION

In the following evaluation, we will compare the data reduction efficiency of the M4 aggregation with state-of-the-art line simplification approaches and with commonly used naive approaches, such as averaging, sampling, and rounding. Therefore, we apply all considered techniques to several real world data sets and measure the resulting visualization quality. For all aggregation-based data reduction operators, which can be expressed using the relational algebra, we also evaluate the query execution performance.

6.1 Real World Time Series Data

We consider three different data sets: the price of a single share on the Frankfurt stock exchange over 6 weeks (700k tuples), 71 minutes from a speed sensor of a soccer ball [22](ball number 8, 7M rows), and one week of sensor data from an electrical power sensor of a semiconductor manufacturing machine [15](sensor MF03, 55M rows). In Figure 11, we show excerpts of these data sets to display the differences in time and value distribution.

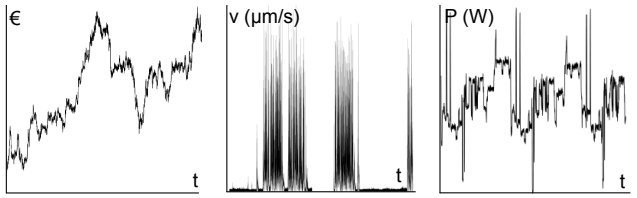


Figure 11: a) financial, b) soccer, c) machine data.

For example, the financial data set (a) contains over 20000 tuples per working day, with the share price changing only slowly over time. In contrast to that the soccer data set (b) contains over 1500 readings per second and is best described as a sequence of bursts. Finally, the machine sensor data (c) at 100Hz constitutes a mixed signal that has time spans of low, high, and also bursty variation.

6.2 Query Execution Performance

In Section 4, we described how to express simple sampling or aggregation-based data reduction operators using relational algebra, including our proposed M4 aggregation. We now evaluate the query execution performance of these different operators. All evaluated queries were issued as SQL queries

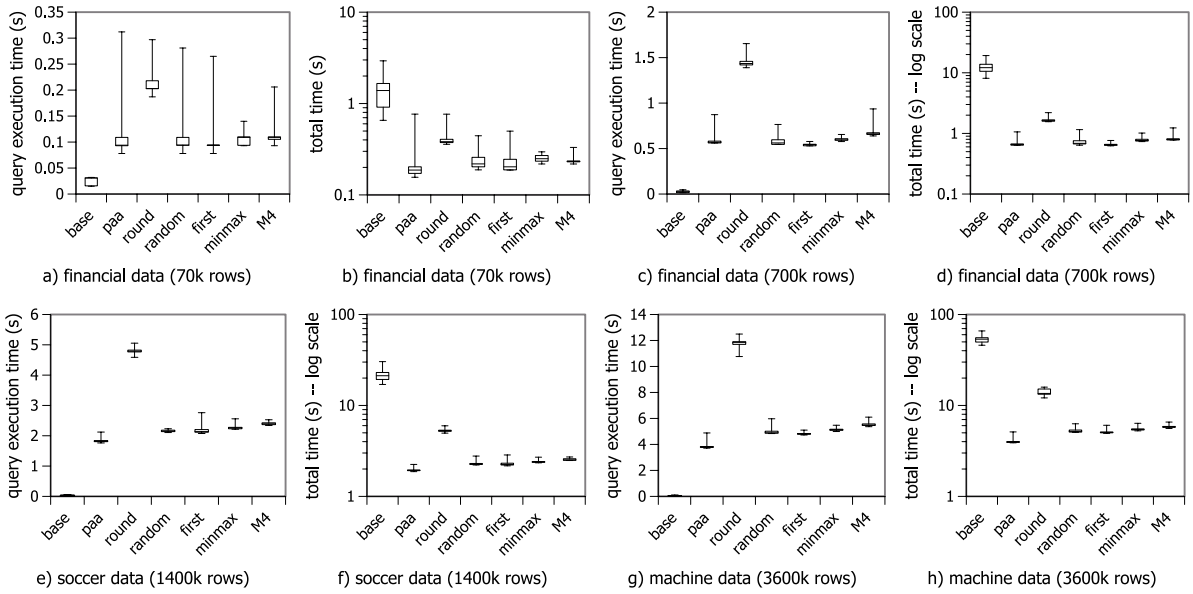


Figure 12: Query performance: (a,b,c,d) financial data, (e,f) soccer data, (g,h) machine data.

via ODBC over a (100Mbit) wide area network to a virtualized, shared SAP HANA v1.00.70 instance, running in an SAP data center at a remote location.

The considered queries are: 1) a baseline query that selects all tuples to be visualized, 2) a PAA-query that computes up to $4 \cdot w$ average tuples, 3) a two-dimensional rounding query that selects up to $w \cdot h$ rounded tuples, 4) a stratified random sampling query that selects $4 \cdot w$ random tuples, 5) a systematic sampling query that selects $4 \cdot w$ first tuples, 6) a MinMax query that selects the two *min* and *max* tuples from $2 \cdot w$ groups, and finally 7) our M4 query selecting all four extrema from w groups. Note that we adjusted the group numbers to ensure a fair comparison at similar data reduction rates, such that any reduction query can at most produce $4 \cdot w$ tuples. All queries are parametrized using a width $w = 1000$, and (if required) a height $h = 200$.

In Figure 12, we plot the corresponding *query execution times* and *total times* for the three data sets. The *total time* measures the time from issuing the query to receiving all results at the SQL client. For the financial data set, we first selected three days from the data (70k rows). We ran each query 20 times and obtain the query execution times, shown in Figure 12a. The fastest query is the baseline query, as it is a simple selection without additional operators. The other queries are slower, as they have to compute the additional data reduction. The slowest query is the rounding query, because it groups the data in two dimensions by w and h. The other data reduction queries only require one horizontal grouping. Comparing these execution times with the total times in Figure 12b, we see the baseline query losing its edge in query execution, ending up one order of magnitude slower than the other queries. Even for the low number of 70k rows, the baseline query is dominated by the additional data transport time. Regarding the resulting total times, all data reduction queries are on the same level and manage to stay below one second. Note that the M4 aggregation does not have significantly higher query execution times and total times than the other queries. The observations are similar when selecting 700k rows (30 days) from the financial data set (Figure 12c and 12d). The aggregation-based queries, including M4, are overall one order of magnitude faster than the baseline at a negligible increase of query execution time.

Our measurements show very similar results when running the different types of data reduction queries on the soccer and machine data sets. We requested 1400k rows from the soccer data set, and 3600k rows from the machine data set. The results in Figure 12(e–h) again show an improvement of total time by up to one order of magnitude.

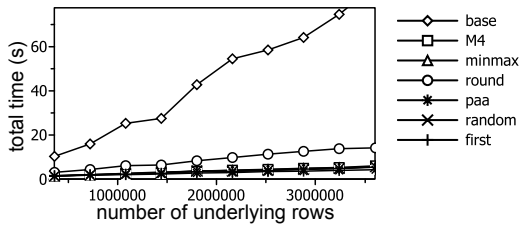


Figure 13: Performance with varying row count.

We repeated all our tests, using a Postgres 8.4.11 RDBMS running on an Xeon E5-2620 with 2.00GHz, 64GB RAM, 1TB HDD (no SSD) on Red Hat 6.3, hosted in the same data center as the HANA instances. All data was served from a RAM disk. The Postgres working memory was set to 8GB. In Figure 13, we show the exemplary results for the soccer data set, plotting the total time for increasing, requested time spans, i.e., increasing number of underlying rows. We again observe the baseline query heavily depending on the limited network bandwidth. The aggregation-based approaches again perform much better. We made comparable observations with the finance data set and the machine data set.

The results of both the Postgres and the HANA system show that the baseline query, fetching all rows, mainly depends on the database-outgoing network bandwidth. In con-

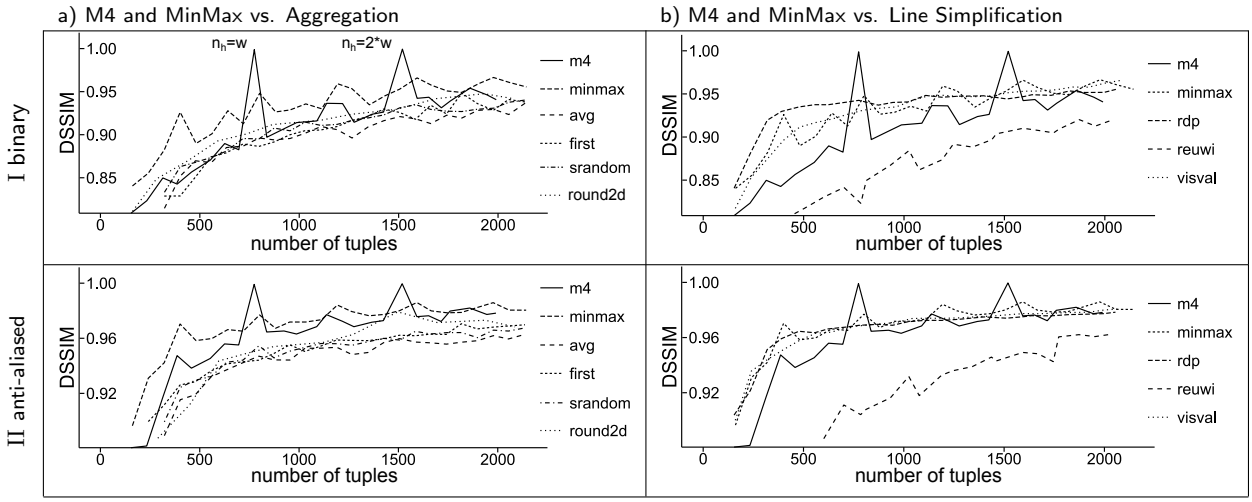


Figure 14: Data efficiency of evaluated techniques, showing DSSIM over data volume.

trast, the size of the result sets of all aggregation-based queries for any amount of underlying data is more or less constant and below $4 \cdot w$. Their total time mainly depends on the database-internal query execution time. This evaluation also shows that our proposed M4 aggregation is equally fast as common aggregation-based data reduction techniques. M4 can reduce the time the user has to wait for the data by one order of magnitude in all our scenarios, and still provide the correct tuples for high quality line visualizations.

6.3 Visualization Quality and Data Efficiency

We now evaluate the robustness and the data efficiency regarding the achievable visualization quality. Therefore, we test M4 and the other aggregation techniques using different numbers of horizontal groups n_h. We start with $n_h = 1$ and end at $n_h = 2.5 \cdot w$. Thereby we want to select at most $10 \cdot w$ rows, i.e., twice as much data as is actually required for an error-free two-color line visualization. Based on the reduced data sets we compute an (approximating) visualization and compare it with the (baseline) visualization of the original data set. All considered visualizations are drawn using the open source Cairo graphics library (cairographics.org). The distance measure is the DSSIM, as motivated in Section 5. The underlying original time series of the evaluation scenario are 70k tuples (3 days) from the financial data set. The related visualization has $w = 200$ and $h = 50$. In the evaluated scenario, we allow the number of groups n_h to be different than the width w of the visualization. This will show the robustness of our approach. However, in a real implementation, the engineers have to make sure that $n_h = w$ to achieve the best results. In addition to the aggregation-based operators, we also compare our approach with three different line simplification approaches, as described in Section 5. We use the Reumann-Wikham algorithm (reuwi) [24] as representative for sequential line simplification, the top-down Ramer-Douglas-Peucker (RDP) algorithm [6, 14], and the bottom-up Visvalingam-Whyatts (visval) algorithm [26]. The RDP algorithm, does not allow setting a desired data reduction ratio, thus we precomputed the minimal ϵ that would produce a number of tuples proportional to the considered n_h.

In Figure 14, we plot the measured, resulting visualization quality (DSSIM) over the resulting number of tuples of each different groupings $n_h = 1$ to $n_h = 2.5 \cdot w$ of an applied data reduction technique. For readability, we cut off all low quality results with $DSSIM < 0.8$. The lower the number of tuples and the higher the DSSIM, the more data efficient is the corresponding technique for the purpose of line visualization. The Figures 14aI and 14bI depict these measures for binary line visualizations and the Figures 14aII and 14bII for anti-aliased line visualizations. We now observe the following results.

Sampling and Averaging operators (*avg*, *first*, and *srandom*) select a single aggregated value per (horizontal) group. They all show similar results and provide the lowest DSSIM. As discussed in Section 4, they will often fail to select the tuples that are important for line rasterization, i.e., the *min*, *max*, *first*, and *last* tuples that are required to set the correct inner-column and inter-column pixels.

2D-Rounding requires an additional vertical grouping into n_v groups. We set $n_v = w/h \cdot n_h$ to a have proportional vertical grouping. The average visualization quality of 2D-rounding is higher than that of averaging and sampling.

MinMax queries select $min(v)$ and $max(v)$ and the corresponding timestamps per group. They provide very high DSSIM values already at low data volumes. On average, they have a higher data efficiency than all aggregation based techniques, (including M4, see Figure 14a), but are partially surpassed by line simplification approaches (see Figure 14b).

Line Simplification techniques (*RDP* and *visval*) on average provide better results than the aggregation-based techniques (compare Figures 14a and 14b). As seen previously [25], top-down (*RDP*) and bottom-up (*visval*) algorithms perform much better than the sequential ones (*reuwi*). However, in context of rasterized line visualizations they are surpassed by M4 and also MinMax at $n_h = w$ and $n_h = 2 \cdot w$. These techniques often miss one of the *min*, *max*, *first*, or *last* tuples, because these tuples must not necessarily comply to the geometric distance measures used for line simplification, as described in Section 5.

M4 queries select $min(v)$, $max(v)$, $min(t)$, $max(t)$ and the corresponding timestamps and values per group. On average M4 provides a visualization quality of $DSSIM > 0.9$

but is usually below MinMax and the line simplification techniques. However, at $n_h = w$, i.e., at any factor k of w, M4 provides perfect (error-free) visualizations. Any grouping with $n_h = k \cdot w$ and $k \in \mathbb{N}^+$ also includes the *min*, *max*, *first*, and *last* tuples for $n_h = w$.

Anti-aliasing. The observed results for binary visualization (Figures 14I) and anti-aliased visualizations (Figures 14II) are very similar. The absolute DSSIM values for anti-aliased visualizations are even better than for binary ones. This is caused by a single pixel error in a binary visualization implying a full color swap from one extreme to the other, e.g., from back (0) to white (255). Pixel errors in anti-aliased visualization are less distinct, especially in overplotted areas, which are common for high-volume time series data. For example, a *missing line* will often result in a small increase in brightness, rather than a complete swap of full color values, and an additional *false line* will result in a small decrease in brightness of a pixel.

6.4 Evaluation of Pixel Errors

A visual result of M4, MinMax, RDP, and averaging (PAA), applied to 400 seconds (40k tuples) of the machine data set, is shown in Figure 15. We use only 100×20 pixels for each visualization to reveal the pixel errors of each operator. M4 thereby also presents the error-free baseline image. We marked the pixel errors for MinMax, RDP, and PAA; black represents additional pixels and white the missing pixels compared to the base image.

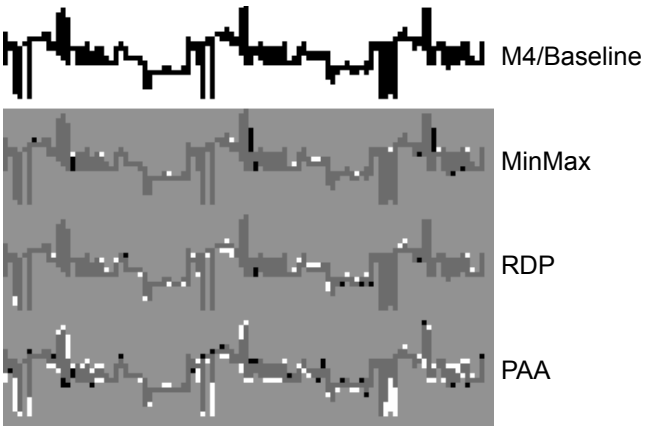


Figure 15: Projecting 40k tuples to 100x20 pixels.

We see how MinMax draws very long, false connection lines (right of each of the three main positive spikes of the chart). MinMax also has several smaller errors, caused by the same effect. In this regard, RDP is better, as the distance of the not selected points to a long, false connection line is also very high, and RDP will have to split this line again. RDP also applies a slight averaging in areas where the time series has a low variance, since the small distance between low varying values also decreases the corresponding measured distances. The most pixel errors are produced by the PAA-based data reduction, mainly caused by the averaging of the vertical extrema. Overall, MinMax results in 30 false pixels, RDP in 39 false pixels, and PAA in over 100 false pixels. M4 stays error-free.

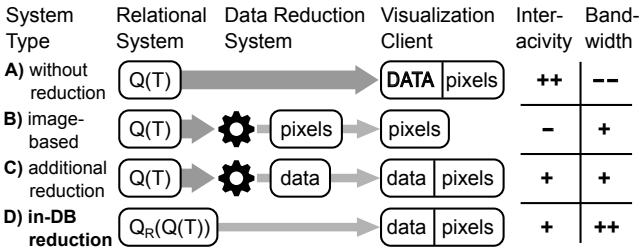


Figure 16: Visualization system architectures.

6.5 Data Reduction Potential

Let us now go back to the motivating example in Section 1. For the described scenario, we expected 100 users trying to visually analyze 12 hours of sensor data, recorded at 100Hz. Each user has to wait for over 4 Million rows of data until he or she can examine the sensor signal visually. Assuming that the sensor data is visualized using a line chart that relies on an M4-based aggregation, and that the maximum width of a chart is $w = 2000$ pixels. Then we know that M4 will at most select $4 \cdot w = 8000$ tuples from the time series, independent of the chosen time span. The resulting maximum amount of tuples, required to serve all 100 users with error-free line charts, is $100_{\text{users}} \cdot 8000 = 800000$ tuples; instead of previously 463 million tuples. As a result, in this scenario we achieve a data reduction ratio of over $1 : 500$.

7. RELATED WORK

In this section, we discuss existing visualization systems and provide an overview of related data reduction techniques, discussing the differences to our approach.

7.1 Visualization Systems

Regarding visualization-related data reduction, current state-of-the-art visualization systems and tools fall into three categories. They (A) do not use any data reduction, or (B) compute and send images instead of data to visualization clients, or (C) rely on additional data reduction outside of the database. In Figure 16, we compare these systems to our solution (D), showing how each type of system applies and reduces a relational query Q on a time series relation T. Note that thin arrows indicate low-volume data flow, and thick arrows indicate that raw data needs to be transferred between the system's components or to the client.

Visual Analytics Tools. Many visual analytics tools are systems of type A that do not apply any visualization-related data reduction, even though they often contain state-of-the-art (relational) data engines [28] that could be used for this purpose. For our visualization needs, we already evaluated four common candidates for such tools: Tableau Desktop 8.1 (tableausoftware.com), SAP Lumira 1.13 (saplumira.com), QlikView 11.20 (clickview.com), and Datawatch Desktop 12.2 (datawatch.com). But none of these tools was able to quickly and easily visualize high-volume time series data, having 1 million rows or more. Since all tools allow working on data from a database or provide a tool-internal data engine, we see a great opportunity for our approach to be implemented in such systems. For brevity, we cannot provide a more detailed evaluation of these tools.

Client-Server Systems. The second system type B is commonly used in web-based solutions, e.g., financial websites like Yahoo Finance (finance.yahoo.com) or Google Finance (google.com/finance). Those systems reduce the data volumes by generating and caching raster images, and sending those instead of the actual data for most of their smaller visualizations. Purely image-based systems usually provide poor interactivity and are backed with a complementary system of type C, implemented as a rich-client application that allows exploring the data interactively. Systems B and C usually rely on additional data reduction or image generation components *between* the data engine and the client. Assuming a system C that allows arbitrary non-aggregating user queries Q, they will regularly need to transfer large query results from the database to the external data reduction components. This may consume significant system-internal bandwidth and heavily impact the overall performance, as data transfer is one of the most costly operations.

Data-Centric System. Our visualization system (type D) can run expensive data reduction operations directly inside the data engine and still achieve the same level of interactivity as provided by rich-client visualization systems (type C). Our system rewrites the original query Q, using additional the data reduction operators, producing a new query Q_R. When executing the new query, the data engine can then jointly optimize all operators in one single query graph, and the final (physical) operators can all directly access the shared in-memory data without requiring additional, expensive data transfer.

7.2 Data Reduction

In the following we give an overview on common data reduction methods and how they are related to visualizations.

Quantization. Many visualization systems explicitly or implicitly reduce continuous times-series data to discrete values, e.g., by generating images, or simply by rounding the data, e.g., to have only two decimal places. A rounding function is a surjective function and does not allow correct reproduction of the original data. In our system we also consider lossy, rounding-based reduction, and can even model it as relational query, facilitating a data-centric computation.

Time Series Representation. There are many works on time series representations [9], especially for the task of data mining [11]. The goal of most approaches is, similar to our goal, to obtain a much smaller representation of a complete time series. In many cases, this is accomplished by splitting the time series (horizontally) into equidistant or distribution-based time intervals and computing an aggregated value (average) for each interval [18]. Further reduction is then achieved by mapping the aggregates to a limited alphabet, for example, based on the (vertical) distribution of the values. The results are, e.g., character sequences or lists of line segments (see Section 5) that approximate the original time series. The validity of a representation is then tested by using it in a data mining tasks, such as time-series similarity matching [29]. The main difference of our approach is our focus on relational operators and our incorporation of the semantics of the visualizations. None of the existing approaches discussed the related aspects of line rasterization that facilitate the high quality and data efficiency of our approach.

Offline Aggregation and Synopsis. Traditionally, aggregates of temporal business data in OLAP cubes are very coarse grained. The number of aggregation levels is limited, e.g., to years, months, and days, and the aggregation functions are restricted, e.g., to *count*, *avg*, *sum*, *min*, and *max*. For the purpose of visualization, such pre-aggregated data might not represent the raw data very well, especially when considering high-volume time series data with a time resolution of a few milliseconds. The problem is partially mitigated by the provisioning of (hierarchical or amnesic) data synopsis [7, 13]. However, synopsis techniques again rely on common time series dimensionality reduction techniques [11], and thus are subject to approximation errors. In this regard, we see the development of a visualization-oriented data synopsis system that uses the proposed M4 aggregation to provide error-free visualizations as a challenging subject to future work.

Online Aggregation and Streaming. Even though this paper focuses on aggregation of static data, our work was initially driven by the need for interactive, real-time visualizations of high-velocity streaming data [16]. Indeed, we can apply the M4 aggregation for online aggregation, i.e., derive the four extremum tuples in $\mathcal{O}(n)$ and in a single pass over the input stream. A *custom M4* implementation could scan the input data for the extremum *tuples* rather than the extremum *values*, and thus avoid the subsequent join, as required by the *relational M4* (see Section 4.2).

Data Compression. We currently only consider data reduction at the *application level*. Any additional *transport-level* data reduction technique, e.g., data packet compression or specialized compression of numerical data [20, 11], is complementary to our data reduction.

Content Adaptation. Our approach is similar to content adaptation in general [21], which is widely used for images, videos, and text in web-based systems. Content adaptation is one of our underlying ideas that we extended towards a relational approach, with a special attention of the semantics of line visualizations.

Statistical Approaches. Statistical databases [1] can serve approximate results. They serve highly reduced approximate answers to user queries. Nevertheless, these answers cannot very well represent the raw data for the purpose of line visualization, since they apply simple random or systematic sampling, as discussed in Section 4. In theory, statistical databases could be extended with our approach, to serve for example *M4* or *MinMax* query results as approximating answers.

7.3 Visualization-Driven Data Reduction

The usage of visualization parameters for data reduction has been partially described by Burtini et al. [3], where they use the *width* and *height* of a visualization to define parameters for some time series compression techniques. However, they describe a client-server system of type C (see Figure 16), applying the data reduction outside of the database. In our system, we push all data processing down to database by means of query rewriting. Furthermore, they use an average aggregation with w groups, i.e., only $1 \cdot w$ tuples, as baseline and do not consider the visualization of the original time series. Thereby, they overly simplify the actual problem and the resulting line charts will lose important detail in the vertical extrema. They do not appropriately discuss the semantics of rasterized line visualizations.

8. CONCLUSION

In this paper, we introduced a visualization-driven query rewriting technique that facilitates a data-centric time series dimensionality reduction. We showed how to enclose all visualization-related queries to an RDBMS within additional data reduction operators. In particular, we considered aggregation-based data reduction techniques and described how they integrate with the proposed query-rewriting.

Focusing on line charts, as the predominant form of time series visualizations, our approach exploits the semantics of line rasterization to drive the data reduction of high-volume time series data. We introduced the novel M4 aggregation that selects the *min*, *max*, *first*, and *last* tuples from the time spans corresponding to the pixel columns of a line chart. Using M4 we were able to reduce data volumes by two orders of magnitude and latencies by one order of magnitude, while ensuring pixel-perfect line visualizations.

In the future, we want to extend our current focus on line visualizations to other forms of visualization, such as bar charts, scatter plots and space-filling visualizations. We aim to provide a general framework for data-reduction that considers the rendering semantics of visualizations. We hope that this in-depth, interdisciplinary database and computer graphics research paper will inspire other researchers to investigate the boundaries between the two areas.

9. REFERENCES

[1] S. Agarwal, A. Panda, B. Mozafari, A. P. Iyer, S. Madden, and I. Stoica. Blink and it's done: Interactive queries on very large data. *PVLDB*, 5(12):1902–1905, 2012.

[2] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[3] G. Burtini, S. Fazackerley, and R. Lawrence. Time series compression for adaptive chart generation. In *CCECE*, pages 1–6. IEEE, 2013.

[4] J. X. Chen and X. Wang. Approximate line scan-conversion and antialiasing. In *Computer Graphics Forum*, pages 69–78. Wiley, 1999.

[5] David Salomon. *Data Compression*. Springer, 2007.

[6] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica Journal*, 10(2):112–122, 1973.

[7] Q. Duan, P. Wang, M. Wu, W. Wang, and S. Huang. Approximate query on historical stream data. In *DEXA*, pages 128–135. Springer, 2011.

[8] S. G. Eick and A. F. Karr. Visual scalability. *Journal of Computational and Graphical Statistics*, 11(1):22–43, 2002.

[9] P. Esling and C. Agon. Time-series data mining. *ACM Computing Surveys*, 45(1):12–34, 2012.

[10] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. SAP HANA Database-Data Management for Modern Business Applications. *SIGMOD Record*, 40(4):45–51, 2012.

[11] T. Fu. A review on time series data mining. *EAAI Journal*, 24(1):164–181, 2011.

[12] T. Fu, F. Chung, R. Luk, and C. Ng. Representing financial time series based on data point importance. *EAAI Journal*, 21(2):277–300, 2008.

[13] S. Gandhi, L. Foschini, and S. Suri. Space-efficient online approximation of time series data: Streams, amnesia, and out-of-order. In *ICDE*, pages 924–935. IEEE, 2010.

[14] J. Hershberger and J. Snoeyink. *Speeding up the Douglas-Peucker line-simplification algorithm*. University of British Columbia, Department of Computer Science, 1992.

[15] Z. Jerzak, T. Heinze, M. Fehr, D. Gröber, R. Hartung, and N. Stojanovic. The DEBS 2012 Grand Challenge. In *DEBS*, pages 393–398. ACM, 2012.

[16] U. Jugel and V. Markl. Interactive visualization of high-velocity event streams. In *VLDB PhD Workshop*. VLDB Endowment, 2012.

[17] D. A. Keim, C. Panse, J. Schneidewind, M. Sips, M. C. Hao, and U. Dayal. Pushing the limit in visual data exploration: Techniques and applications. *Lecture notes in artificial intelligence*, (2821):37–51, 2003.

[18] E. J. Keogh and Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. In *PAKDD*, pages 122–133. Springer, 2000.

[19] A. Kolesnikov. *Efficient algorithms for vectorization and polygonal approximation*. University of Joensuu, 2003.

[20] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. In *TVCG*, volume 12, pages 1245–1250. IEEE, 2006.

[21] W.-Y. Ma, I. Bedner, G. Chang, A. Kuchinsky, and H. Zhang. A framework for adaptive content delivery in heterogeneous network environments. In *Proc. SPIE, Multimedia Computing and Networking*, volume 3969, pages 86–100. SPIE, 2000.

[22] C. Mutschler, H. Ziekow, and Z. Jerzak. The DEBS 2013 Grand Challenge. In *DEBS*, pages 289–294. ACM, 2013.

[23] P. Przymus, A. Boniewicz, M. Burzańska, and K. Stencel. Recursive query facilities in relational databases: a survey. In *DTA and BSBT*, pages 89–99. Springer, 2010.

[24] K. Reumann and A. P. M. Witkam. Optimizing curve segmentation in computer graphics. In *Proceedings of the International Computing Symposium*, pages 467–472. North-Holland Publishing Company, 1974.

[25] W. Shi and C. Cheung. Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, 43(1):27–44, 2006.

[26] M. Visvalingam and J. Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, 1993.

[27] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[28] R. Wesley, M. Eldridge, and P. Terlecki. An analytic data engine for visualization in tableau. In *SIGMOD*, pages 1185–1194. ACM, 2011.

[29] Y. Wu, D. Agrawal, and A. El Abbadi. A comparison of DFT and DWT based similarity search in timeseries databases. In *CIKM*, pages 488–495. ACM, 2000.