Time Series Data Validity

YUNXIANG SU, Tsinghua University, China
YIKUN GONG, Tsinghua University, China
SHAOXU SONG, Tsinghua University, China

As a key step of data preparation, it is always necessary to first assert the quality of data before conducting any data application. Given a set of constraints, the *validity* measure evaluates the degree of data meeting the constraints, e.g., whether the values are in the specified range or fluctuate drastically over time in a series. It is worth noting that simply counting all the data points in violation to the constraints may over claim the data validity issue. Following the minimum change criteria in data repairing, we propose to study the minimum number of data points that need to be changed in order to satisfy the constraints, or equivalently, the maximum rate of data that can be reserved without change, as the validity measure. To our best knowledge, this is the first study on defining and evaluating time series data validity. We devise algorithms for computing the validity measure in quadratic time and linear space. Remarkably, the validity measure has been deployed and included as a function in SQL statements, in Apache IoTDB, an open-source time series database. The algorithm fully adapts to the LSM-based storage of time series in multiple segments. Extensive experiments over 8 real-world datasets show up to 4 orders of magnitude improvement in time cost compared to the related method SCREEN.

CCS Concepts: • **Information systems** → **Data cleaning**; *Enterprise applications*; • **Applied computing** → **Enterprise data management**.

Additional Key Words and Phrases: data quality, time series data, IoT

ACM Reference Format:
Yunxiang Su, Yikun Gong, and Shaoxu Song. 2023. Time Series Data Validity. *Proc. ACM Manag. Data* 1, 1, Article 85 (May 2023), 26 pages. https://doi.org/10.1145/3588939

1 INTRODUCTION

Time series data are often found with various data quality issues, especially in the Internet of Things (IoT), with frequent device and transmission failures [24]. Analysis over the dirty data is obviously not trusted, e.g., seriously misleading regression [11], classification [16], clustering [23], etc. Asserting data quality does not only indicate unreliable results, but also has business value, e.g., giving a signal of device maintenance (see some case studies in Section 6.7).

Constraints are declared to validate time series data. The value constraints specify whether values are in the specified range, such as temperature range. The speed constraints check whether values fluctuate drastically over time, e.g., GPS trajectory constrained by walking speed [25, 26]. Given a set of constraints, validity measure evaluates to what extent the data satisfy constraints.

This work is supported in part by National Natural Science Foundation of China (62021002, 62072265, 62232005), National Key Research and Development Plan (2021YFB3300500), Beijing National Research Center for Information Science and Technology (BNR2022RC01011), and Alibaba Group through Alibaba Innovative Research (AIR) Program. Shaoxu Song (https://sxsong.github.io/) is the corresponding author.
Authors' addresses: Yunxiang Su, Tsinghua University, Beijing, China, suyx21@mails.tsinghua.edu.cn; Yikun Gong, Tsinghua University, Beijing, China, gyk19@mails.tsinghua.edu.cn; Shaoxu Song, Tsinghua University, Beijing, China, sxsong@tsinghua.edu.cn.

2836-6573/2023/5-ART85
https://doi.org/10.1145/3588939

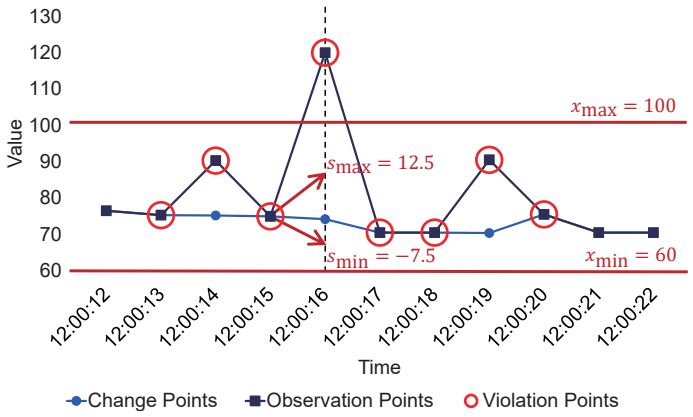


Fig. 1. Value constraints $[x_{\min}, x_{\max}]$ and speed constraints $[s_{\min}, s_{\max}]$ for evaluating data validity

It is worth noting that simply counting all the data points in violation to the constraints may over claim the data validity issue (see example below). Following the minimum change criteria in data repairing [7], we propose to study the minimum number of data points that need to be changed, in order to eliminate violations. That is, the validity measure is defined as the maximum rate of data points that can be reserved without change, to satisfy the constraints. To the best of our knowledge, this is the first study on defining and evaluating validity of time series data.

Example 1. *Figure 1 presents a segment of time series, monitoring the temperature of water tank in a vehicle, from our industrial partner (i.e., the TY-Vehicle data in experiments in Section 6.1).*

The value constraints of water temperature are $[60, 100]$*, denoted by two red lines. It states that the value of each data point should be within the range. Point at time 12:00:16 is thus detected as a violation to the constraints.*

Moreover, the value fluctuation between two points indicates the speed of their value movement, e.g., the values fluctuating from 75 at time 12:00:15 to 120 at time 12:00:16 having speed $\frac{120-75}{16-15} = 45$*. Informally, the speed constraints thus specify the maximum and minimum value fluctuations, e.g., denoted by two red arrows in Figure 1, respectively. For instance, the speed constraints* $[-7.5, 12.5]$*, provided by domain experts, state that the temperature value fluctuation between two points should not be greater than 12.5 for each time unit, and no less than -7.5. Two consecutive points with slope not between these two arrows are detected as violations to the speed constraints. Thus, points at 12:00:15 and 12:00:16, whose slope exceeds the range, are violation points.*

Simply counting all the data points in violation, i.e., 8 red points from time 12:00:13 to 12:00:20, may over claim the data quality issue. Indeed, only 3 points need to change, e.g., to the dots at time 12:00:14, 12:00:16 and 12:00:19, to satisfy the value and speed constraints. Hence, we propose to define the validity as the rate of data points that have no need to change, i.e., $\frac{8}{11}$ *rather than over claimed* $\frac{3}{11}$*.*

1.1 Background on Time Series Data Store

Consider a time series $\mathbf{x}$ of n data points, i.e., $n = \|\mathbf{x}\|$. Let x_i and t_i denote the value and time of the i-th point in the time series. By default, t_i serves as the key of the time series, i.e., any two points (in a segment) should not have the same time.

To support the extremely intensive write loads of IoT data, the Log-Structured Merge-Tree (LSM-Tree) [20] is often employed, e.g., in Apache IoTDB [4], an open-source time series database management system. Figure 2 presents an example of LSM-based storage. A time series is segmented

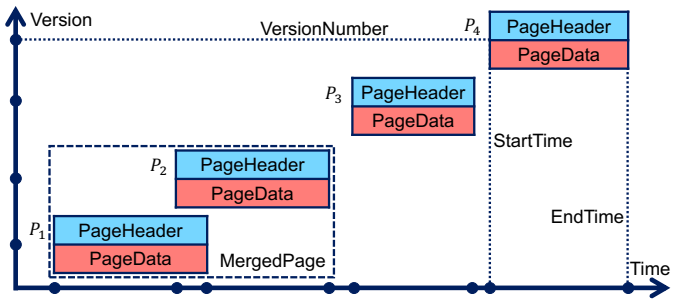


Fig. 2. LSM-based storage of time series in multi-segments

into multiple pages, e.g., P_1 to P_4. Each page consists of PageHeader recording necessary metadata and PageData for time series data (as illustrated in Figure 6 below). The StartTime and EndTime in the PageHeader of each page denote the timestamps of the start and end points in the segment, respectively, e.g., t_1 and t_k for the segment $\mathbf{x}[1:k]$ stored in page P_1. Since each page is written at different time, a unique version number is associated to each page, such as P_4 with version number 4. The larger the version number is, the later the page is written.

It is worth noting that the time intervals, from StartTime to EndTime, of different pages may overlap. The reason is that the arrival of data points could be out-of-order. For example, the start point of P_2 has a timestamp smaller than the end point of P_1, since the former is delayed and written to page P_2 with a higher version number after P_1 stored. Even more complicated, a point in P_1 may be overwritten (updated) by another point in P_2 with the same timestamp. It often occurs in a situation when the connection to a sensor is interrupted and a default value of the sensor is written first in P_1. Once the connection is recovered, the true value sent by the sensor arrives and is stored in P_2 as an update of the point stored in P_1.

1.2 Contribution

Our major contributions in this paper are as follows.

(1) We define the validity measure for time series in Section 2. For LSM-based storage of a time series in multiple segments (pages) in commodity time series databases, unfortunately, the validity of multiple segments could not be directly aggregated, owing to their interacting time ranges.

(2) We devise algorithms for computing the validity measure of a single segment, in quadratic time and linear space, in Section 3. Intuitively, we prove that there must exist one point having no need to change, unless all the data points violate the value constraints. The computation of validity thus departs from this fixed point.

(3) We compute the validity of time series stored in multiple segments in Section 4. Remarkably, it is proved that the validity of segments can be directly aggregated in certain cases, significantly improving efficiency. Moreover, we propose to re-split the segments of time series to enable more such direct aggregations.

(4) We have deployed the validity measure in Apache IoTDB [4], an open-source time series database management system, in Section 5. It is included as a function in SQL statements. We introduce two types of deployment, fully adapting to the LSM-based storage for write intensive loads and enabling pre-computation.

(5) We demonstrate the efficiency of our proposal over real-world datasets in Section 6. It verifies the necessity of data validity assertion for time series.

Table 1. Notations

Symbol	Description
$\mathbf{x}$	time series of $n = \|\mathbf{x}\|$ data points
(t_i, x_i)	time and value of i-th data point in $\mathbf{x}$
s_i	speed of value fluctuation from x_{i-1} to x_i
$[x_{\min}, x_{\max}]$	constraints of minimum and maximum values
$[s_{\min}, s_{\max}]$	constraints of minimum and maximum speeds
validity($\mathbf{x}$)	validity of time series $\mathbf{x}$

Finally, we discuss related work in Section 7 and conclude the paper in Section 8. Table 1 lists the frequently used notations.

2 PRELIMINARY

In this section, we first introduce two types of constraints for evaluating time series data validity in Section 2.1. The validity measure is then formally defined in Section 2.2.

2.1 Constraints for Validity

We consider two types of constraints, on values and speeds, which can either be specified by domain experts or profiled from data [25].

2.1.1 Value Constraints. As introduced in Section 1, value constraints, specifying the ranges of values such as temperature, are the most common constraints in practice. The *value constraints*, in a form of $[x_{\min}, x_{\max}]$, state that all data points i should have

$$x_{\min} \le x_i \le x_{\max}.$$

For instance, $x_5 = 120$ at time 12:00:16 in Figure 1 is not in the range of value constraints $[x_{\min}, x_{\max}] = [60, 100]$, i.e., a violation.

Value constraints are often given by domain experts to specify the value range requirements. For example, in monitoring the temperature of water tank in a vehicle, the value constraints of water temperature should be $[60, 100]$. Exceeding this range will be regarded as an error observation (the engine is not possible to run). Therefore, we use value constraints to guide validity evaluation.

2.1.2 Speed Constraints. Data fluctuation is another issue to consider, often constrained as well [25]. Let s_i denote the speed of value fluctuation from point $i-1$ to i, defined by $s_i = \frac{x_i - x_{i-1}}{t_i - t_{i-1}}$. The *speed constraints* $[s_{\min}, s_{\max}]$ state that for each data point i, its speed s_i should be within the range

$$s_{\min} \le \frac{x_i - x_{i-1}}{t_i - t_{i-1}} \le s_{\max}.$$

For instance, x_5 at time 12:00:16 in Figure 1 together with x_4 having speed $s_5 = \frac{120-75}{16-15} = 45$ violate the speed constraints $[s_{\min}, s_{\max}] = [-7.5, 12.5]$.

It is worth noting that the speed constraints are declared on consecutive points $i-1$ and i. According to [33], they are sufficient to guarantee any two points satisfying the speed constraints, i.e., $\forall i, j$, $s_{\min} \le \frac{x_j - x_i}{t_j - t_i} \le s_{\max}$. Therefore, the speed constraints declared on consecutive points are competent for the validity measure.

Speed constraints are also prevalent. For example, consider the traveling miles of a train (not the speed) as the values in the time series. The speed constraints declare that the values of miles should not increase faster than the maximum traveling speed of a train.

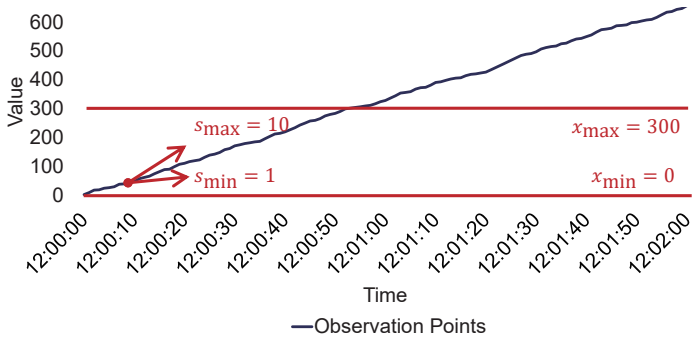


Fig. 3. Inconsistent constraints with $s_{\min} > 0$

2.1.3 Consistent Constraints. Unfortunately, the mistakenly specified constraints could be inconsistent [8]. That is, there does not exist any time series data instance (of size $n \geq 2$ for defining speed) that can satisfy the constraints, e.g., specifying some speed constraints with $s_{\max} < s_{\min}$. It becomes more complicated when value and speed constraints interact with each other. Evaluating data validity w.r.t. the inconsistent constraints is obviously meaningless. Therefore, we assume below the given value and speed constraints having $x_{\min} \leq x_{\max}$, $s_{\min} \leq 0 \leq s_{\max}$, which are always consistent (a simple instance is $x_i = x_{\min}$ for all points i).

Note that the assumption $s_{\min} \leq 0 \leq s_{\max}$ is necessary. Without such an assumption, the constraints could be contradictory, i.e., no data can satisfy the constraints. For instance, in Figure 3, the speed constraints with $s_{\min} > 0$ require the data values always increasing, while the value constraints need the values not exceeding $x_{\max}$. For a time series with length larger than the value domain size, it is impossible to satisfy such constraints, i.e., keep on increasing without exceeding $x_{\max}$. Nevertheless, the proposed algorithm, designed for two constraints, can be extended to handle some other general cases as presented in Section 6.5.

2.1.4 Applicable Scenarios. We consider value and speed constraints, since they are widely observed, e.g., walking speed of GPS trajectory, daily limit of stock prices [25]. Such constraints are also considered in our industrial partners, e.g., the temperature of water tank in a vehicle (in TY) should not fluctuate significantly in a short period, the values of miles should not increase faster than the maximum traveling speed of a train (in CRRC). In short, the constraints are usually applicable to values and speeds with normal distribution, whereas expected sudden increase/decrease (like sales during holiday season) unfortunately cannot be handled.

2.2 Definition of Validity

As illustrated in Example 1, counting all the data points in violation to the constraints may over claim the data validity issue. Following the minimum change criteria in data repairing [7], we propose to study the minimum number of data points that need to be changed in order to satisfy the constraints of data validity. Let $\mathbf{x}'$ be a possible change of $\mathbf{x}$. We consider the number of data points that are changed in $\mathbf{x}'$, i.e.,

$$\Delta(\mathbf{x}, \mathbf{x}') = \|\{x_i \mid x_i \neq x'_i, 1 \leq i \leq n\}\| \tag{1}$$

where $n = \|\mathbf{x}\|$. The cardinality-minimal cost [7] w.r.t. the value and speed constraints studied in this paper is as follows.

Definition 1 (Cardinality-Minimal Cost). *We denote $\delta(\mathbf{x})$ the minimum number of data points in $\mathbf{x}$ that have to be changed in order to satisfy the value and speed constraints.*

$$\begin{aligned} \delta(\mathbf{x}) = \min \quad & \Delta(\mathbf{x}, \mathbf{x}') \\ s.t. \quad & s_{\min} \leq \frac{x'_i - x'_{i-1}}{t_i - t_{i-1}} \leq s_{\max}, && 1 < i \leq n \\ & x_{\min} \leq x'_i \leq x_{\max}, && 1 \leq i \leq n \end{aligned}$$

2.2.1 Alternatives of Inconsistency Measures. Deletion operation cost $\mathcal{I}_{\mathcal{R}}$ [18] measures the minimal deletion operations that make the remaining data satisfy the constraints. Intuitively, deleting all changed points in $\mathbf{x}'$ will make the remaining points meet the value and speed constraints, i.e., equal to the cardinality-minimal cost.

Proposition 1. *Given value constraints $[x_{\min}, x_{\max}]$ and speed constraints $[s_{\min}, s_{\max}]$, for a time series $\mathbf{x}$, the cardinality-minimal cost $\delta(\mathbf{x})$ is equal to the deletion operation cost $\mathcal{I}_{\mathcal{R}}$ [18] (a.k.a. the cost of S-repair [17]) w.r.t. the value and speed constraints.*

Moreover, motivated by probabilistic functional dependencies [30], the violation count measures the number of data points that violate value and speed constraints. Each point violating the value constraints or each pair of consecutive points violating the speed constraints is indeed a minimal inconsistent subset, i.e., a problematic fact [18].

Proposition 2. *Given value constraints $[x_{\min}, x_{\max}]$ and speed constraints $[s_{\min}, s_{\max}]$, for a time series $\mathbf{x}$, the violation count is equal to the problematic inconsistency measure $\mathcal{I}_{\mathrm{P}}$ [18, 28].*

In addition to the aforesaid equivalent measures, we evaluate other alternative inconsistency measures, such as MI Shapley Inconsistency [18] and distance-minimal cost [9], in Section 6.6.

2.2.2 Validity Measure. The validity is thus defined on $\delta(\mathbf{x})$, considering the maximum number of data points that do not have to change in order to meet the requirement of the value and speed constraints.

Definition 2 (Time Series Data Validity). *Given value constraints $[x_{\min}, x_{\max}]$ and speed constraints $[s_{\min}, s_{\max}]$, the validity measure of a time series $\mathbf{x}$ is the maximum ratio of data points that can be reserved, or equivalently, 1− the minimum ratio of data points that should be changed (not reserved) to satisfy the constraints.*

$$\mathsf{validity}(\mathbf{x}) = 1 - \frac{\delta(\mathbf{x})}{\|\mathbf{x}\|}$$

Given the interacting value and speed constraints, determining the minimum number of changed points (or the maximum number of reserved) could be costly.

Example 2 (Example 1 continued). *For the possible change $\mathbf{x}'$ in Figure 1, with points changed at time 12:00:14, 12:00:16 and 12:00:19, we have cost $\Delta(\mathbf{x}, \mathbf{x}') = 3$. It is indeed the minimum change that can satisfy the constraints. The cardinality-minimal cost is thus $\delta(\mathbf{x}) = 3$, leading to the validity of this time series $\mathsf{validity}(\mathbf{x}) = 1 - \frac{3}{11} = \frac{8}{11}$.*

The satisfaction of value and speed constraints depends on what the values are changed to. For example, in Figure 1, by changing three points at time 12:00:14, 12:00:16 and 12:00:19 to the blue values, both constraints are satisfied, i.e., with values in the range of red lines and speeds in the included angle of red arrows. Indeed, the validity measure counts the minimum number of points whose values need to be changed. That is, by changing such a number of points, there must exist a changed time series with values satisfying the constraints. While the validity measure ensures the existence of valid value changes, it is not the focus of this study to determine the specific values changed to (known as another data repairing problem [25]).

2.3 Validity over LSM-based Store

Let $\mathbf{x}[1:n]$ be the time series by merging the multiple segments $\mathbf{x}[1:k_1], \mathbf{x}[k_1{+}1:k_2], \ldots, \mathbf{x}[k_m{+}1:n]$ stored in pages. For those points with the same timestamp in different segments, only the one with the largest version number will be reserved in $\mathbf{x}[1:n]$.

Definition 3 (Multi-Segment Time Series Data Validity). *Given value constraints* $[x_{\min}, x_{\max}]$ *and speed constraints* $[s_{\min}, s_{\max}]$*, the validity for multiple segments* $\mathbf{x}[1:k_1], \mathbf{x}[k_1{+}1:k_2], \ldots, \mathbf{x}[k_m{+}1:n]$ *is the validity of the corresponding merged time series* $\mathbf{x}[1:n]$, $\text{validity}(\mathbf{x}[1:n])$.

To compute the validity of multi-segment time series, a natural idea is thus to read all the segments from disks and merge them as a whole time series in memory. We devise an efficient algorithm for computing the validity of a time series, in Section 3.

Obviously, loading all data in memory is costly in space, while computing validity of a long time series is CPU intensive. Intuitively, it will be promising to (pre)compute the validity for each segment and aggregate directly the validity, rather than merging the data. Unfortunately, the validity of multiple segments might not be directly aggregated, owing to their interacting time ranges, such as the aforesaid P_1 and P_2 in Figure 2. Nevertheless, we identify that in certain cases in Proposition 7, the validity measures of different segments can be computed separately and aggregated. Therefore, we propose to re-split the segments of time series to enable more such efficient aggregations, in Section 4. The aforesaid ideas lead to two different deployment ways of the validity measure, in the LSM-based Apache IoTDB, in Section 5.

3 SINGLE-SEGMENT TIME SERIES VALIDITY

In this section, we focus on the validity of a time series in one segment, as in Definition 2. Intuitively, when all the data points violate the value constraints and need to change with $\delta(\mathbf{x}) = \|\mathbf{x}\|$, it directly has $\text{validity}(\mathbf{x}) = 0$. Therefore, we only need to study the remaining cases that at least one data point has no need to change. Remarkably, we can split the time series on the unchanged data point (Section 3.1) and solve them separately (Section 3.2).

3.1 Splitting into Sub-Problems

While the special case above considers all the data points violating the value constraints, directly having $\text{validity}(\mathbf{x}) = 0$, Lemma 3 below studies its complement, i.e., having at least one point satisfying the value constraints. It shows that at least one data point does not need to be changed, to meet the constraints. By this unchanged point, Proposition 4 splits the problem into two sub-problems for computing the validity.

Lemma 3. *For a time series* $\mathbf{x}$*, if there exists at least one data point with value satisfying the value constraints, then we have* $\delta(\mathbf{x}) < \|\mathbf{x}\|, \quad \text{validity}(\mathbf{x}) > 0$.

Let m denote the unchanged point, having $x_m = x'_m$. The cost function in Formula (1) can be rewritten as

$$\Delta(\mathbf{x}, \mathbf{x}') = \|\{x_i \mid 1 \le i < m, x_i \ne x'_i\}\| + \|\{x_j \mid m < j \le n, x_j \ne x'_j\}\|$$

on the remaining $n-1$ points. It splits the time series into two parts. The former one has unchanged end point m, while the latter one starts with the unchanged point m. In this sense, we further study two special cases of time series, with unchanged end point and unchanged start point, respectively.

Definition 4 (Fixed-End Cost). *We denote $\delta_e(\mathbf{x})$ the cardinality-minimal cost without changing the end point of $\mathbf{x}$.*

$$\begin{aligned}\delta_e(\mathbf{x}) = \min \quad & \Delta(\mathbf{x}, \mathbf{x}') \\ s.t. \quad & s_{\min} \le \frac{x'_i - x'_{i-1}}{t_i - t_{i-1}} \le s_{\max}, && 1 < i \le n \\ & x_{\min} \le x'_i \le x_{\max}, && 1 \le i \le n \\ & x_n = x'_n\end{aligned}$$

For the special case of fixed-end, the end point n of $\mathbf{x}$ will not be changed. Likewise, we consider another special case of fixed-start, where the value x_1 of the start point will be unchanged.

Definition 5 (Fixed-Start Cost). *We denote $\delta_s(\mathbf{x})$ the cardinality-minimal cost without changing the start point of $\mathbf{x}$.*

$$\begin{aligned}\delta_s(\mathbf{x}) = \min \quad & \Delta(\mathbf{x}, \mathbf{x}') \\ s.t. \quad & s_{\min} \le \frac{x'_j - x'_{j-1}}{t_j - t_{j-1}} \le s_{\max}, && 1 < j \le n \\ & x_{\min} \le x'_j \le x_{\max}, && 1 \le j \le n \\ & x_1 = x'_1\end{aligned}$$

We are now ready to split the problem into two sub-problems by the unchanged point m. Lemma 3 only states that there exists one unchanged point without indicating which point it is. Therefore, we need to traverse all the possible m of unchanged point (satisfying the value constraints).

Proposition 4. *For a time series $\mathbf{x}$ with at least one data point satisfying the value constraints, its $\delta(\mathbf{x})$ can be computed by solving two sub-problems $\delta_e(\mathbf{x}[1:m])$ and $\delta_s(\mathbf{x}[m:n])$, i.e.,*

$$\delta(\mathbf{x}) = \min_{1 \le m \le n} \delta_e(\mathbf{x}[1:m]) + \delta_s(\mathbf{x}[m:n])$$

where $\mathbf{x}[1:m]$ is the prefix of $\mathbf{x}$ with length m, $\mathbf{x}[m:n]$ is the suffix of $\mathbf{x}$ with length $n-m+1$, and x_m satisfies the value constraints $x_{\min} \le x_m \le x_{\max}$.

Therefore, the problem of computing $\delta(\mathbf{x})$ can be split as solving $\delta_e(\mathbf{x}[1:m])$ and $\delta_s(\mathbf{x}[m:n])$ for all data points m satisfying the value constraints.

Example 3 (Example 2 continued). *Let the data point at time 12:00:18 be the unchanged point m, i.e., x_7 in Figure 4 satisfying the value constraints. As illustrated, it splits the time series into two sub-problems, $\delta_e(\mathbf{x}[1:7])$ with fixed-end point x_7 and $\delta_s(\mathbf{x}[7:11])$ with fixed-start point x_7, which will be solved in the following examples. Similar split should also be applied to other points satisfying the value constraints, e.g., data point at time 12:00:17 with $m = 6$ will also be considered as unchanged for another split.*

3.2 Solving Fixed-End Sub-Problem

To solve $\delta_e(\mathbf{x}[1:m])$ with fixed end point m, we note that all the previous points of m must have speeds with point m satisfying the speed constraints, referring to the discussion in Section 2.1.2. Intuitively, we may find another unchanged point u before m, and recursively solve the problem $\delta_e(\mathbf{x}[1:u])$ with fixed end u in Proposition 5. If no point u is found unchanged before m, i.e., the edge case, we can directly obtain the validity $\delta_e(\mathbf{x}[1:m])$ in Proposition 6.

First, let u be the point closest to m that satisfies the constraints. It is easy to see that all the data points from $u+1$ to $m-1$ need to be changed, given the unchanged x_m, with cost $m-u-1$.

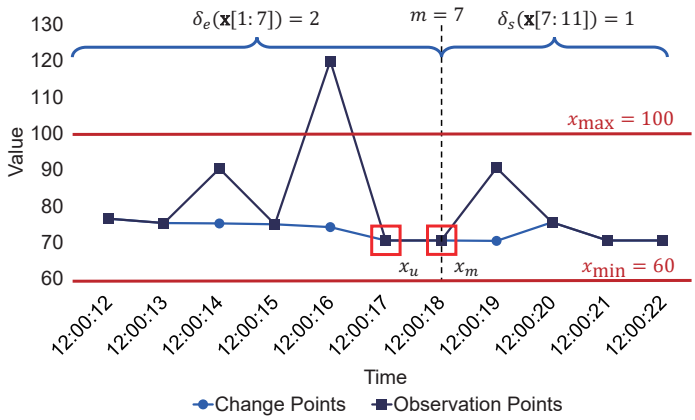


Fig. 4. Example of solving sub-problems

Moreover, for some other u that satisfies the constraints, if all the data points from $u+1$ to $m-1$ are also changed, we still have the cost $m-u-1$. Otherwise, for the unchanged point, say u' between u and m, it has already been recursively considered as aforesaid. In this sense, by traversing all the possible u that could be unchanged, we can compute $\delta_e(\mathbf{x}[1:m])$.

Proposition 5. *For a time series $\mathbf{x}[1:m]$ with fixed end point m, i.e., $x_{\min} \le x_m \le x_{\max}$, if there exists a point u having*

$$1 \le u < m,$$
$$x_{\min} \le x_u \le x_{\max},$$
$$s_{\min} \le \frac{x_m - x_u}{t_m - t_u} \le s_{\max},$$

then $\delta_e(\mathbf{x}[1:m])$ can be recursively computed by $\delta_e(\mathbf{x}[1:m]) = \min_{1 \le u < m}(\delta_e(\mathbf{x}[1:u]) + m - u - 1)$.

For the edge case that no such unchanged point u exists, we have to change all the points before m with cost $m-1$.

Proposition 6. *For a time series $\mathbf{x}[1:m]$ with fixed end point m, i.e., $x_{\min} \le x_m \le x_{\max}$, if there does **not** exist any point u having*

$$1 \le u < m,$$
$$x_{\min} \le x_u \le x_{\max},$$
$$s_{\min} \le \frac{x_m - x_u}{t_m - t_u} \le s_{\max},$$

then $\delta_e(\mathbf{x}[1:m])$ can be directly given by $\delta_e(\mathbf{x}[1:m]) = m-1$.

Referring to Propositions 5 and 6, we devise Algorithm 1, which recursively computes the fixed-end cost $\delta_e(\mathbf{x}[1:m])$. It first initializes $\delta_e(\mathbf{x}[1:m]) := m-1$ according to Proposition 6. For x_m, we traverse all the possibly unchanged u, $1 \le u < m$, in Line 2. If x_u and x_m satisfy the requirements in Proposition 5, $\delta_e(\mathbf{x}[1:m])$ is updated in Line 6. Line 4 ensures that each $\delta_e(\mathbf{x}[1:u])$ is computed at most once. By computing $\delta_e(\mathbf{x}[1:u])$ at most once for each possibly unchanged point u, the algorithm runs in $O(n)$ time and space.

Algorithm 1 Fixed-end cost

Input: time series $\mathbf{x}[1:m]$, value constraints $[x_{\min}, x_{\max}]$ and speed constraints $[s_{\min}, s_{\max}]$
Output: fixed-end cost $\delta_e(\mathbf{x}[1:m])$

1: $\delta_e(\mathbf{x}[1:m]) := m-1$
2: **for** $u := m-1$ to 1 **do**
3: **if** $x_{\min} \le x_u \le x_{\max}$ **and** $s_{\min} \le \frac{x_m - x_u}{t_m - t_u} \le s_{\max}$ **then**
4: **if** $\delta_e(\mathbf{x}[1:u])$ is not computed **then**
5: recursively compute $\delta_e(\mathbf{x}[1:u])$ with fixed-end
6: $\delta_e(\mathbf{x}[1:m]) := \min(\delta_e(\mathbf{x}[1:u]) + m - u - 1, \delta_e(\mathbf{x}[1:m]))$
7: **return** $\delta_e(\mathbf{x}[1:m])$

Example 4 (Example 3 continued). *To compute $\delta_e(\mathbf{x}[1:7])$ for fixed-end point $m=7$ at time 12:00:18 in Figure 4, Algorithm 1 recursively considers other possibly unchanged point u. For the edge cases, we can simply get $\delta_e(\mathbf{x}[1:1]) = 0, \delta_e(\mathbf{x}[1:2]) = 0$. For the possibly unchanged point at time 12:00:14, x_2 need to be changed, having $\delta_e(\mathbf{x}[1:3]) = 1$. Following the same line of checking all the possibly unchanged point u before m, we obtain $\delta_e(\mathbf{x}[1:7]) = \delta_e(\mathbf{x}[1:6]) = \delta_e(\mathbf{x}[1:4]) + 1 = \delta_e(\mathbf{x}[1:2]) + 1 + 1 = 2$. That is, two points 3 and 5 at time 12:00:14 and 12:00:16 need to change for the fixed-end point 7.*

Following the same line of solving the fixed-end problem, $\delta_s(\mathbf{x}[m:n])$ for fixed-start problem can also be recursively computed in the same way.

3.3 Algorithm for Single-Segment Time Series

After solving the two sub-problems, we are now ready to compute $\delta(\mathbf{x})$. Algorithm 2 presents the pseudo-code for computing $\delta(\mathbf{x})$ and finally validity$(\mathbf{x})$. It considers all the possibly unchanged x_m from Line 3 to 7, and returns the minimum $\delta_e(\mathbf{x}[1:m]) + \delta_s(\mathbf{x}[m:n])$ according to Proposition 4. Again, by computing $\delta_e(\mathbf{x}[1:u])$ and $\delta_s(\mathbf{x}[u:n])$ at most once for each possibly unchanged point u, the algorithm runs in $O(n^2)$ time and $O(n)$ space.

Algorithm 2 Single-segment time series validity

Input: time series $\mathbf{x}$, value constraints $[x_{\min}, x_{\max}]$ and speed constraints $[s_{\min}, s_{\max}]$
Output: the validity measure validity$(\mathbf{x})$

1: $n := \|\mathbf{x}\|$
2: $\delta(\mathbf{x}) := n$
3: **for** $m := 1$ to n **do**
4: **if** $x_{\min} \le x_m \le x_{\max}$ **then**
5: compute $\delta_e(\mathbf{x}[1:m])$ with fixed-end
6: compute $\delta_s(\mathbf{x}[m:n])$ with fixed-start
7: $\delta(\mathbf{x}) := \min((\delta_e(\mathbf{x}[1:m]) + \delta_s(\mathbf{x}[m:n])), \delta(\mathbf{x}))$
8: **return** $1 - \frac{\delta(\mathbf{x})}{n}$

Example 5 (Example 4 continued). *Similar to Example 4, a variation of Algorithm 1 computes $\delta_s(\mathbf{x}[7:11]) = 1$ for fixed-start point $m=7$ at time 12:00:18 in Figure 4. It follows $\delta_e(\mathbf{x}[1:7]) + \delta_s(\mathbf{x}[7:11]) = 2+1 = 3$ for unchanged point $m=7$. Referring to Proposition 4, Algorithm 2 calculates $\delta_e(\mathbf{x}[1:m]) + \delta_s(\mathbf{x}[m:11])$ for all the possibly unchanged point m, i.e., all the points except x_5 not satisfying the value constraints. The minimum one is $\delta(\mathbf{x}) = 3$, leading to* validity$(\mathbf{x}) = 1 - \frac{3}{11} = \frac{8}{11}$.

3.4 Arbitrarily Bad Case

Lemma 3 considers the case that there exists at least one data point with value satisfying the value constraints. Though it may violate the speed constraints, such a point is proved having no need to be changed, i.e., a lower bound of validity > 0. It leads to the fixed-point solution (without changing the aforesaid point) in Section 3.2. For the other cases, i.e., all the points violate the value constraints, they need to be all changed. Algorithm 2 can handle such arbitrarily bad cases, always false in Line 4 and thus outputting validity 0, i.e., exact solutions (not approximation).

4 MULTI-SEGMENT TIME SERIES VALIDITY

In this section, we consider the more general case of time series stored in multiple segments, i.e., $\mathbf{x}[1 : k_1], \mathbf{x}[k_1 + 1 : k_2], \ldots, \mathbf{x}[k_m + 1 : n]$ in Definition 3. To compute the validity of the whole time series, a straightforward idea is to merge all the segments as a whole, introduced in Section 2.3, and apply the algorithms proposed in Section 3 for single segment time series.

Rather than costly merging the data of segments as a long series for computation, we identify some special segments whose validity could be computed individually and aggregated directly in Proposition 7, in Section 4.1. For the segments that cannot be aggregated, we propose to re-split the time series, so that the efficient aggregation may be applicable to the new segments, in Section 4.2.

4.1 Aggregating Validity of Segments

Intuitively, for those segments without overlapping time intervals and unchanged start/end points, their changes will not affect each other. In this sense, the validity measures could be directly aggregated. It is worth noting that this direct aggregation enables not only the segment re-split solution of a long series, but also the reuse of the pre-computed segment validity, as presented below.

Proposition 7. *For two segments* $\mathbf{x}[1 : k]$ *and* $\mathbf{x}[k + 1 : n]$*, if*

$$t_k < t_{k+1}, \tag{2}$$

$$x_k = x'_k, \tag{3}$$

$$x_{k+1} = x'_{k+1}, \tag{4}$$

$$s_{\min} \leq \frac{x_{k+1} - x_k}{t_{k+1} - t_k} \leq s_{\max}, \tag{5}$$

then $\mathsf{validity}(\mathbf{x}[1 : n])$ *can be directly aggregated by*

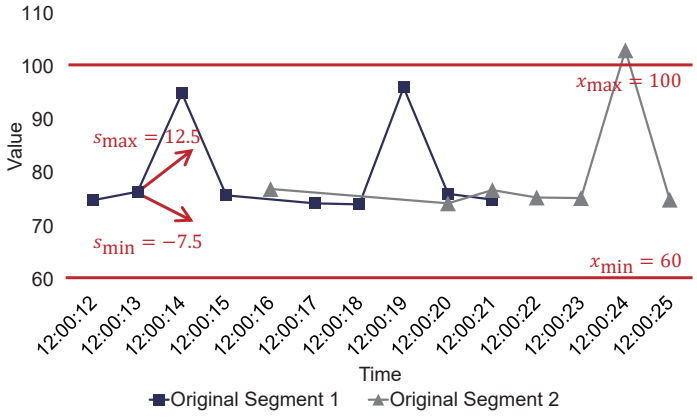
$$\mathsf{validity}(\mathbf{x}[1 : n]) = \mathsf{validity}(\mathbf{x}[1 : k])\frac{k}{n} + \mathsf{validity}(\mathbf{x}[k + 1 : n])\frac{n - k}{n}, \tag{6}$$

where $x_k = x'_k$ *denotes that* k *is a fixed end point in calculating* $\mathsf{validity}(\mathbf{x}[1 : k])$*, and similarly* $x_{k+1} = x'_{k+1}$ *means a fixed start point for* $\mathsf{validity}(\mathbf{x}[k + 1 : n])$*.*

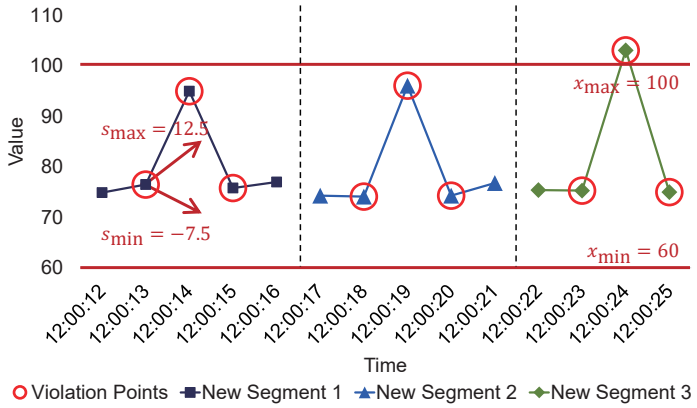
In practice, 90% time series segments can be directly aggregated, leading to significantly lower time cost in Section 6. The reason is that out-of-order arrivals and errors do not occur frequently.

4.2 Re-splitting Segments for Aggregation

For segments that cannot apply Proposition 7, we propose to re-split the segments so that they can be aggregated efficiently. To use Proposition 7, segments are expected to satisfy two aspects of conditions, (1) non-overlapping time intervals in Formula 2, and (2) satisfying constraints without change in Formulas 3-5, i.e., non-violation.



(a) Two original segments of a time series



(b) Re-split into 3 new segments

Fig. 5. Re-split a time series of two segments into three

First, the segments $\mathbf{x}[1 : k_1], \mathbf{x}[k_1 + 1 : k_2], \ldots, \mathbf{x}[k_m + 1 : n]$, with possibly overlapping time intervals such as $t[k_1 + 1] < t[k_1]$, should be re-split into segments $\mathbf{x}[1 : k'_1], \mathbf{x}[k'_1 + 1 : k'_2], \ldots, \mathbf{x}[k'_{m'} + 1 : n]$ without overlapping, i.e., $t[k'_1] < t[k'_1 + 1]$, etc.

Intuitively, for two consecutive non-violation points, if they are not changed in computing validity, the constraints in Formulas 3-5 are satisfied. In this sense, it is a candidate place of splitting segments for possible validity aggregation. Intuitively, the more the non-violation points appear together, the more unlikely the points need to change in computing validity. Thereby, we may split in the middle for every c consecutive non-violation points.

Once the segments are re-split, we compute validity separately on each new segment, calling the single segment Algorithm 2. The overall validity can be obtained by aggregating (or merging) consecutive segments, instead of computing validity over the whole series.

Example 6. *Figure 5(a) present two segments of time series, where Segment 2 has a higher version number than Segment 1. To compute the validity of the overall time series, a naive method is to first merge Segment 1 and Segment 2. As introduced in Section 2.3, for those points with the same timestamp, e.g., at 12:00:20 or 12:00:21, only the points with the higher version number will be reserved in the*

merged time series, i.e., the gray ones in Segment 2. Validity is then computed on the merged time series, similar to Example 5.

To re-split the segments, we consider each place of two consecutive non-violation points, i.e., (12:00:16, 12:00:17) and (12:00:21, 12:00:22) for $c = 2$. By splitting in the middle of every two consecutive non-violation points, we obtain 3 new segments in Figure 5(b). After computing the validity of all 3 segments, Proposition 7 is found applicable, i.e., unchanged points at 12:00:16, 12:00:17, 12:00:21, 12:00:22. The overall validity is thus aggregated by $\frac{0.8\times5}{14} + \frac{0.8\times5}{14} + \frac{0.75\times4}{14} = \frac{11}{14}$.

4.3 Algorithm for Multi-Segment Time Series

With the validity aggregation and segment re-split, we are now ready to efficiently calculate validity for multi-segment time series in Algorithm 3. It first merges overlapping segments in Line 2, and re-splits the segments in Line 6, according to Section 4.2. The validity of the new split segment is of course not pre-computed and needs online computing, by either calling Algorithm 2 or directly obtained during splitting.

Once the validity of each re-split segment is computed, together with the pre-computed validity of segments without splitting, the validity of all segments is obtained. Line 8 further merges non-aggregate segments, which cannot meet the conditions in Proposition 7. Finally, the validity of all the remaining segments can be directly aggregated, in Line 13.

It takes linear time to merge and re-split segments. Let κ be the maximum number of segments and ℓ be the maximum length of a segment. Referring to the complexity of Algorithm 2, the time and space costs for computing the validity of a segment are $O(\ell^2)$ and $O(\ell)$. Merging non-aggregate segments in Line 8 occurs at most κ times. Algorithm 3 thus runs in $O(n + \ell^2\kappa)$ time and $O(\ell)$ space.

Algorithm 3 Multi-segment time series validity

Input: segments of time series $\mathbf{x}$ ordered by start time, value constraints $[x_{\min}, x_{\max}]$ and speed constraints $[s_{\min}, s_{\max}]$
Output: the validity measure validity($\mathbf{x}$)

1: **for** each pair of consecutive segments $\mathbf{x}[i:k]$, $\mathbf{x}[k+1:j]$ **do**
2: **if** $t_k \geq t_{k+1}$ **then**
3: $\mathbf{x}[i:j] := \mathbf{x}[i:k] \cup \mathbf{x}[k+1:j]$
4: replace $\mathbf{x}[i:k]$ and $\mathbf{x}[k+1:j]$ by $\mathbf{x}[i:j]$ to continue
5: **for** each segment $\mathbf{x}[i:j]$ not pre-computed **do**
6: split $\mathbf{x}[i:j]$ into segments and compute their validity
7: **for** each pair of consecutive segments $\mathbf{x}[i:k]$, $\mathbf{x}[k+1:j]$ **do**
8: **if** $x_k \neq x'_k$ **or** $x_{k+1} \neq x'_{k+1}$ **or** $\frac{x_{k+1}-x_k}{t_{k+1}-t_k} < s_{\min}$ **or** $\frac{x_{k+1}-x_k}{t_{k+1}-t_k} > s_{\max}$ **then**
9: $\mathbf{x}[i:j] := \mathbf{x}[i:k] \cup \mathbf{x}[k+1:j]$
10: compute validity($\mathbf{x}[i:j]$)
11: replace $\mathbf{x}[i:k]$ and $\mathbf{x}[k+1:j]$ by $\mathbf{x}[i:j]$ to continue
12: **for** each segment $\mathbf{x}[i:k]$ **do**
13: aggregate validity($\mathbf{x}[i:k]$) to validity($\mathbf{x}$) by Formula (6)
14: **return** validity($\mathbf{x}$)

5 SYSTEM DEPLOYMENT

The validity measure has been deployed and included as a function in Apache IoTDB [4], an open-source time series database management system. Users can call the function in a SQL statement.

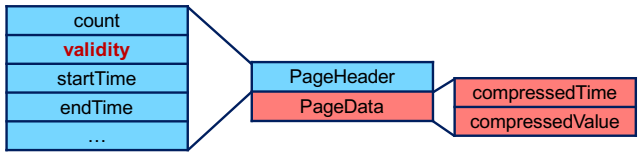


Fig. 6. Page in TsFile

Table 2. Datasets

Name	# series	# data points	Time intervals
CBMI-Engine	12	162,219	Non-uniform
CRRC-Train	5,746	1,157,062	Non-uniform
CSSC-Ship	78	652,491	Non-uniform
GW-WindTurbine	35	1,254,608	Non-uniform
TY-Vehicle	26	1,162,238	Non-uniform
WH-Chemistry	54	1,100,000	Uniform
UCI-Energy	27	19,735	Uniform
UCI-PAMAP2	52	375,908	Uniform

```
select validity(s0) from root.d0 where time > 2017-11-01T00:08:00
```

It returns the validity of time series s0 in the device of root.test.d0, for the specified time range.

Remarkably, we design two ways of deployment for different scenarios, i.e., UDF based deployment without pre-computation for write intensive loads in Section 5.1 and TsFile based deployment with pre-computation for efficient query processing in Section 5.2.

5.1 UDF Deployment without Pre-computation

In this deployment, the system merges all the pages of queried time series, e.g., all the 4 pages in Figure 2 are merged as a whole. Then, Algorithm 2 for single segment time series applies to the merged time series to compute the validity. This UDF implementation is favored in write intensive loads, since no additional computation (compared to the TsFile deployment below) is necessary in the data write phase. However, by merging the whole time series in memory, both the space and time costs of validity query are high.

5.2 TsFile Deployment with Pre-computation

Another deployment is based on TsFile, the files managed in Apache IoTDB, where the pre-computed validity measure is stored for reuse. Figure 6 illustrates the structure of a page in TsFile. In addition to the necessary statistics, such as the aforesaid StartTime and EndTime in Figure 2 as well as count of data points stored in the page, we also record the validity of the time series segment. Algorithm 3 is then conducted over the multiple segments (pages) with pre-computed validity, to obtain the validity of the queried time series. Remarkably, the direct validity aggregation of segments not only reduces the space cost of caching the entire time series in memory, but also improves the IO cost by reading only the metadata in PageHeader without loading the PageData. Therefore, as illustrated in the experiments in Section 6, the TsFile deployment with pre-computation shows orders of magnitude improvement in query time compared to the UDF implementation.

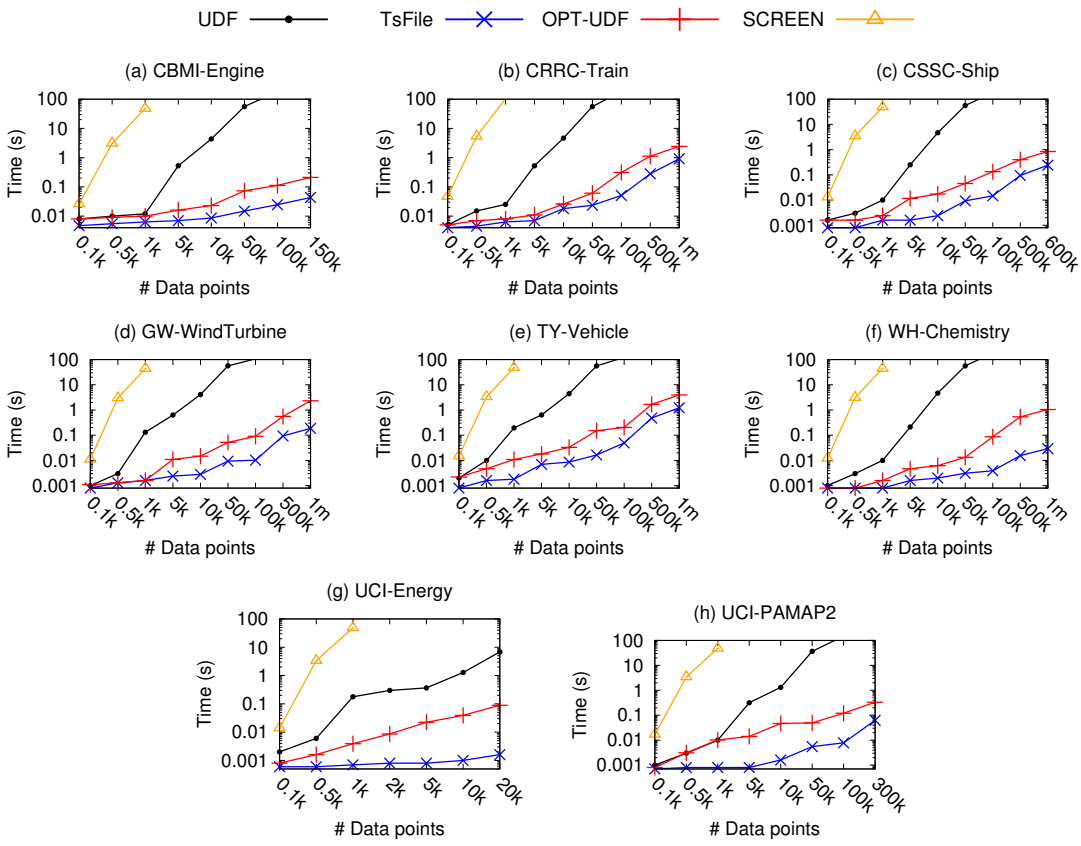


Fig. 7. Validity computation under different data sizes

6 EXPERIMENTS AND CASE STUDIES

In this section, we report the experimental evaluation. The document of the validity measure is available in the website of Apache IoTDB [5]. The source code of UDF and TsFile implementations has been deployed and included in the official GitHub repository of Apache IoTDB [2]. The experiment related code and public data are available at [3]. All these are committed anonymously.

6.1 Experimental Settings

All algorithms are implemented in Java and evaluated on a machine with Intel Core 2 CPU (2.13 GHz) and 8 GB of memory. The experiments run upon Apache IoTDB v0.13, with memory budget 2 GB, page size 1024 and re-split length $c = 100$ by default.

Table 2 lists the real-world datasets employed. While the first 6 ones are property from our industrial partners, we employ two other public UCI datasets [1] for evaluation. The constraints are provided by domain experts of the companies or profiled from data by the three-sigma rule of thumb [21] for UCI datasets. Note that our solution can handle univariate time series with either uniform time intervals or not. As indicated in Table 2, the last three datasets used in experiments are with uniform time intervals.

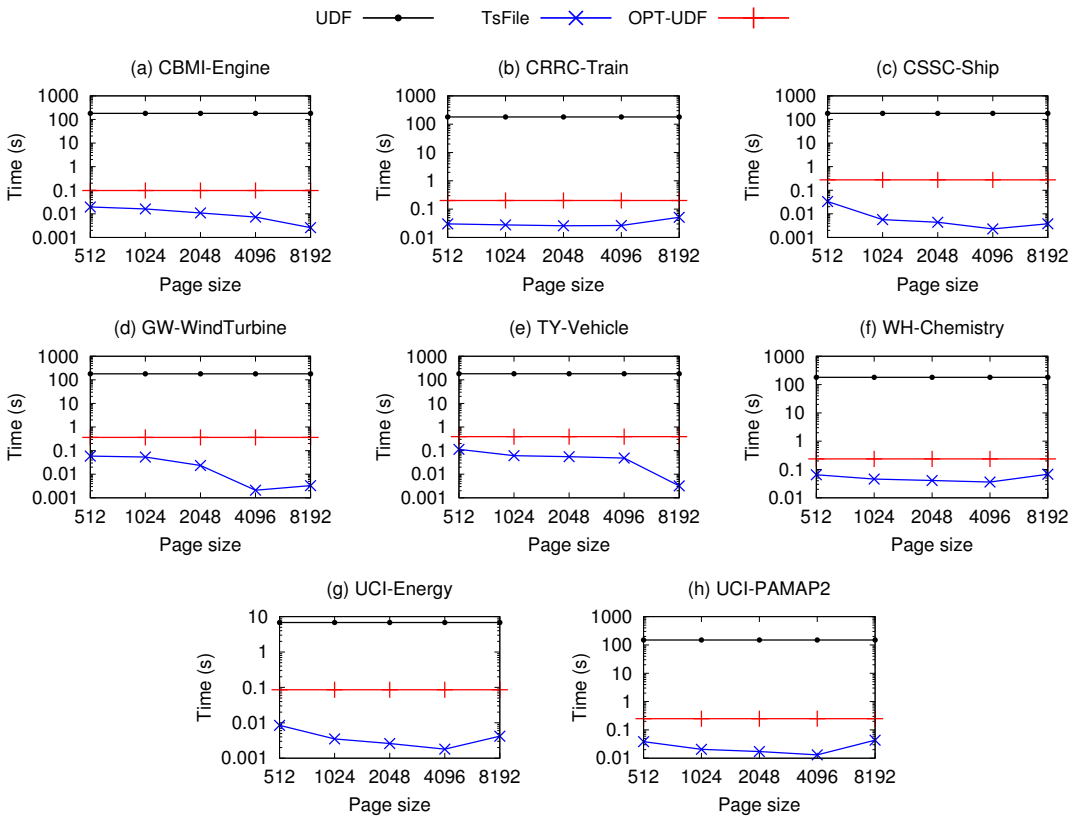


Fig. 8. Validity computation under different page sizes

6.2 Comparison to Related Methods

Since this is the first study on time series data validity, no existing method is available for comparison. Thereby, we adapt the existing time series data repairing method SCREEN [25] as the baseline. It finds a repair with the minimum distance to the original series, by solving as a linear programming (LP) problem. We count the number of points changed in the repair to compute the validity, which however may not be exactly the same as Definition 2.

Figure 7 reports the time cost of validity computation. Owing to the extremely high cost of LP solver, baseline SCREEN performs only with no more than 1,000 points, and thus is omitted in following experiments. Our TsFile based deployment shows up to 4 orders of magnitude improvement compared to SCREEN.

6.3 Evaluation on Splitting Single Segment

It is notable that the "high validity score" property that exhibits in most time-series. In this sense, we may also use "optimistic" splitting described in multi-segment scenario in a single-segment.

6.3.1 UDF with Optimistic Splitting. For UDF, we also evaluate the improvement by optimistic splitting, namely OPT-UDF. As shown in Figure 7, OPT-UDF, splitting the time series into segments, shows much lower time cost than the original UDF, calling Algorithm 2 directly over the whole

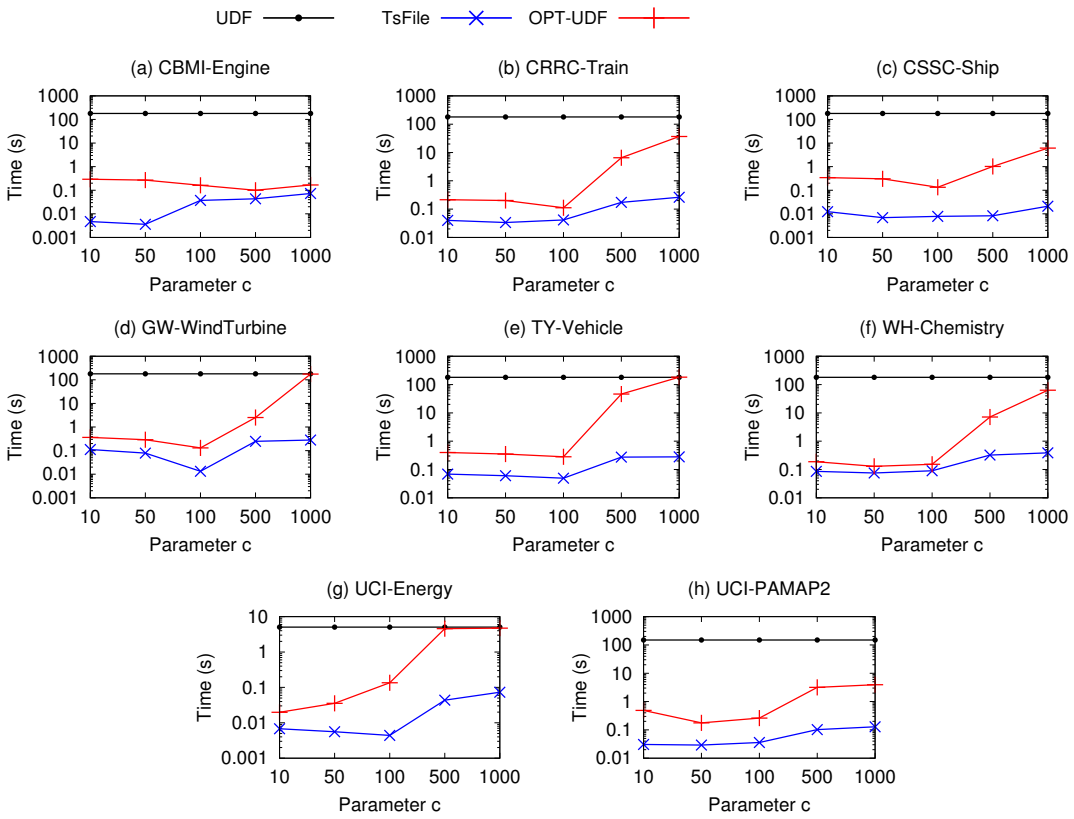


Fig. 9. Validity computation under different re-split c

series. Nevertheless, OPT-UDF without utilizing pre-computation is still not as efficient as the TsFile implementation.

6.3.2 Optimistic Splitting in Pages. It is true that no matter how time-series are segmented in the underlying storage, the algorithm can always split a segment for calculating validity. Figure 8 evaluates the page sizes where the splitting performs. When the page size is too small, there are too many segments split that may not be directly aggregated, leading to high time cost. On the other hand, if the page size is too large, it is costly to recompute for a page. Thereby, a page size of 4096 or 8192 performs best in most tests.

According to the evaluation, the algorithm can automatically decide when to optimistically split. Figure 9 evaluates the length c of consecutive non-violation points for re-split in Section 4.2. A small c leads to many split segments and thus higher processing time. However, a too large c makes segments fail to split by consecutive non-violation points, and could be worse as well. In Figure 9, a length about 100 shows good performance to split in most cases, much smaller than the preferred page size 4096 or 8192 in Figure 8. That is, the algorithm chooses to split all pages.

6.4 Evaluation on Derived Constraints

While some constraints are natural to specify, such as the speed of trains declared on miles, more others could be learnt from data and confirmed by experts. For example, by the three-sigma rule of

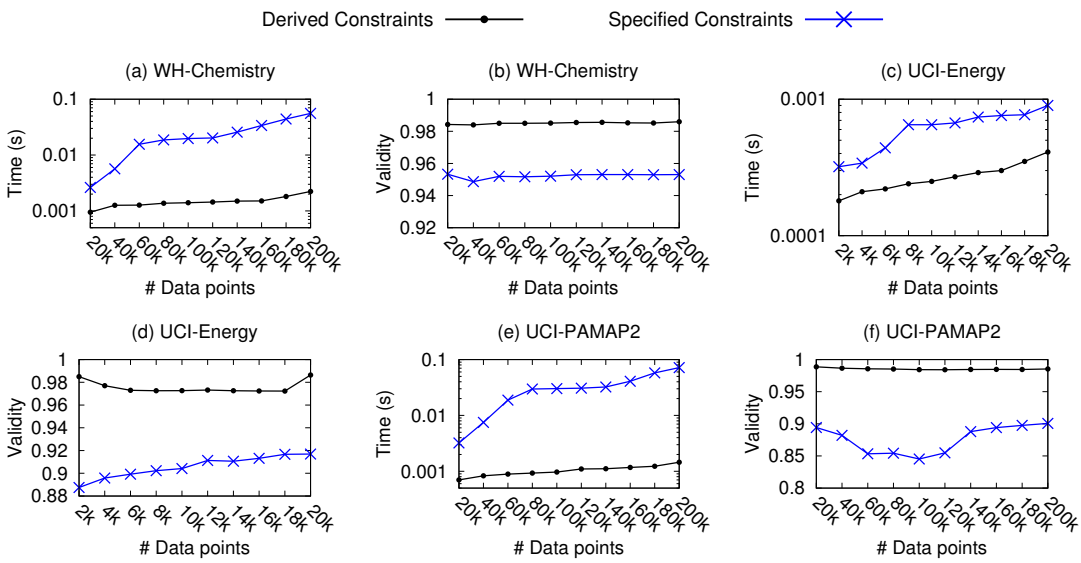


Fig. 10. Evaluation on derived constraints

thumb [21], we can obtain a range of speeds from the speed distribution. The idea also applies to find value constraints.

In particular, for the special case of identical inter arrival time t, some speed constraints can be naturally derived from value constraints, i.e., $[\frac{x_{\min}-x_{\max}}{t}, \frac{x_{\max}-x_{\min}}{t}]$. It means that time series satisfying the value constraints $[x_{\min}, x_{\max}]$ will always satisfy the aforesaid derived speed constraints.

Figure 10 evaluates such interesting cases of derived speed constraints, compared to the manually specified constraints. The experiments are conducted over datasets WH-Chemistry, UCI-Energy and UCI-PAMAP2 with uniform time intervals, as indicated in Table 2. Referring to the aforesaid semantics of the derived speed constraints, it only needs to consider the value constraints such that the derived speed constraints will always be satisfied. Therefore, as shown in Figure 10, the time cost w.r.t. the derived constraints is much lower than that with the additionally specified constraints. However, the additional speed constraints specify more strict requirements on value fluctuations that are not identified by the derived constraints. That is, the validity is indeed not as high as the one indicated by the derived constraints. The results verify that using only value constraints (with speed constraints derived) is usually not sufficient.

6.5 Evaluation on More General Constraints

While the proposed algorithm is designed for two constraints, it can be extended to handle more general cases. Figures 11 and 13 illustrate different constraints on time ranges and sliding windows in our industrial partners, with experimental evaluation as follows.

6.5.1 Evaluation on different time ranges. If different min/max value/speed constraints are declared in different time-range, we can divide the time series according to the time-range and apply the algorithm separately. Since there is no constraint declared between the boundary points of two time ranges, their changes will not affect each other and thus the validity can be efficiently aggregated. Figure 11 shows an example of declaring different constraints in two time ranges in CBMI-Engine, and Figure 12 reports the experimental results on various number of time ranges.

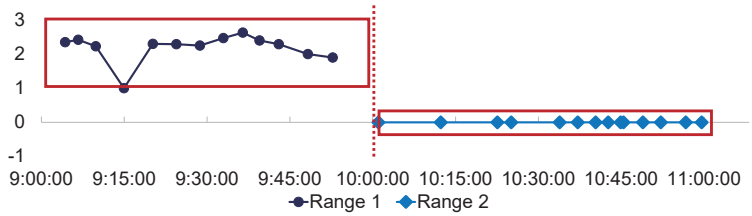


Fig. 11. Different constraints on two time ranges in Rotational Speed in CBMI-Engine

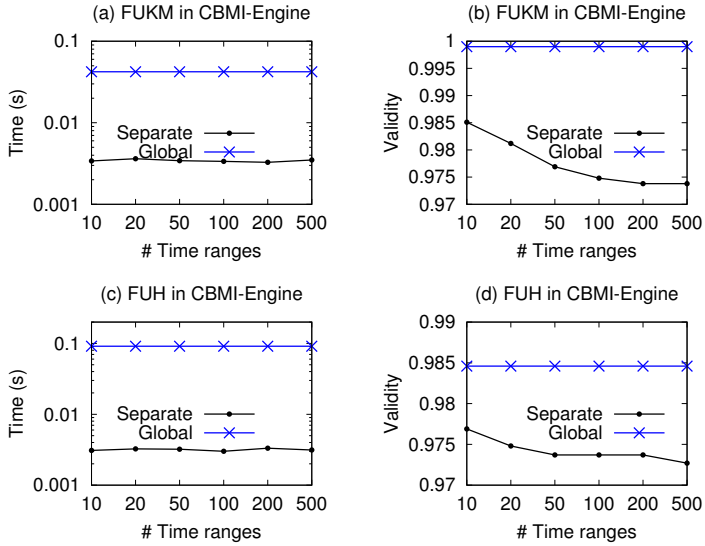


Fig. 12. Evaluation on constraints in different time ranges

As shown in Figure 11 , the vehicle engine works from 9 to 10, and becomes idle at 10, waiting for the next task. For the working and idle stages, validity measures are obtained separately under different constraints, denoted by red rectangles, and then aggregated. As shown in Figure 12 (a) and (c), the time cost (of TsFile implementation) by separate constraints is much lower than that with global constraints, since all time ranges can be directly aggregated. With more strict constraints applied to each time range, the validity by separate constraints is lower than that of global constraints.

6.5.2 Evaluation on sliding windows. For the constraint, e.g., no drastic change in 24 hours, we may declare value constraints in a time range of 24 hours, such that values should not change exceeding the range of $[x_{\min}, x_{\max}]$. Constraints on sliding windows can also be derived as tumbling windows, as in Figure 13. The experimental results on sliding windows are reported in Figure 14.

Each red rectangle in Figure 13 denotes the value constraints in a window of 24 hours, denoting the max/min temperature reported by a wind turbine. It is declared in every 12 hours, i.e., slide size 12. The dash area denotes the intersection of the constraints in the overlapped windows, i.e., the derived constraints that are more strict and satisfy the constraints in both windows. Thereby, in Figure 14, the validity with sliding windows is lower than that of global constraints. The time cost (of TsFile implementation) is generally similar to that in different time ranges in Figure 12.

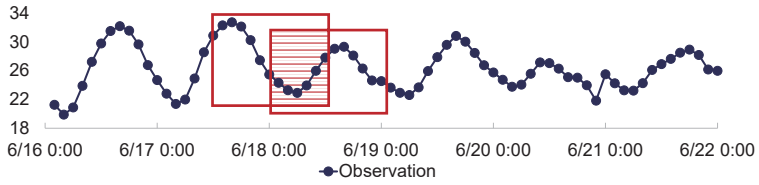


Fig. 13. Different constraints on sliding windows in Temperature in GW-WindTurbine

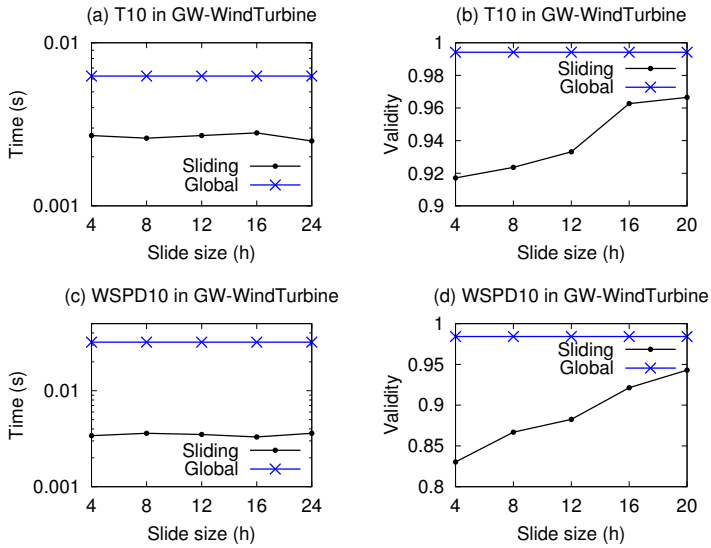


Fig. 14. Evaluation on constraints in sliding windows

6.6 Comparison on Inconsistency Measures

We compare the cardinality-minimal cost [7] employed in this paper with alternative inconsistency measures [9, 18, 28]. Note that for time series with value and speed constraints, the problematic inconsistency measure $\mathcal{I}_{\mathrm{P}}$ [18, 28] is equivalent to the violation count [30] (Proposition 2). Moreover, the cardinality-minimal cost [7] employed in this paper is equivalent to the measure of deletion operation $\mathcal{I}_{\mathcal{R}}$ [18] (Proposition 1). Thereby, we compare with other inconsistency measures such as distance-minimal [9] and MI Shapley Inconsistency [18]. Figures 15, 16 and 17 illustrate the motivation examples of counting the values that need changes, in the application scenarios of our industrial partners. To evaluate the inconsistency measures, Figures 18, 19 and 20 report the experimental comparison in measuring inconsistencies by manually injecting different types of errors. As shown, the employed cardinality-minimal cost is precise, exactly the number of errors injected. Table 3 summarizes the application scenarios, where the employed cardinality-minimal measure is meaningful, while others may over-claim or under-claim data validity issues.

6.6.1 Comparison to Violation Count. Note that violation count may over-claim or under-claim the data validity issue, i.e., neither a lower bound nor an upper bound. For example, 8 red points are counted as violations in Figure 1, while only 3 of them need to be changed to satisfy the constraints, i.e., violation count over-claiming. Similar examples are also observed in Figure 15 of spike errors

Table 3. Inconsistency measures

Measure	Spike Error [25]	Consecutive Error [13]	Shift Error [13]
Violation Count [30]	over-claim	over-claim	under-claim
MI Shapley Inconsistency [18]	over-claim	over-claim	under-claim
Distance-minimal [9]	precise	over-claim	over-claim
Cardinality-minimal [7]	precise	precise	precise

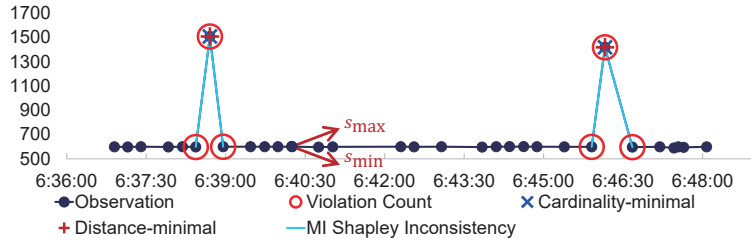


Fig. 15. Spike errors in Rotational Speed in CBMI-Engine

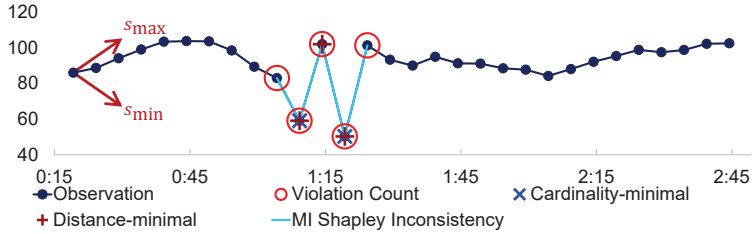


Fig. 16. Consecutive errors in Water Depth in CSSC-Ship

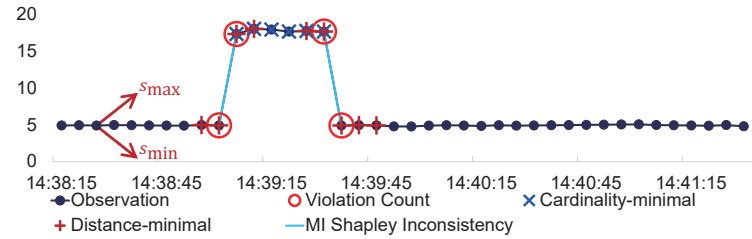


Fig. 17. Shift errors in Sealing Pressure in WH-Chemistry

[25] and Figure 16 of consecutive errors (a.k.a. sequence outliers [13]). Consequently, the violation count is larger than the number of spike and consecutive errors injected in the data, as shown in Figures 18 and 19 of experiments with manually injected errors.

On the other hand, for the case of shift error [13], i.e., a segment of consecutive points integrally shifting values for an amount by mistake, as illustrated in Figure 17, only the start and end points of the shifted segment are detected as violations. However, we need to change all the points in

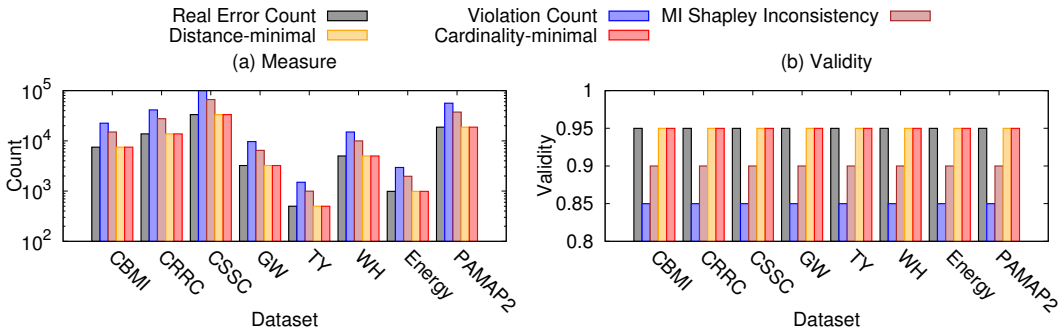


Fig. 18. Inconsistency measures for spike errors

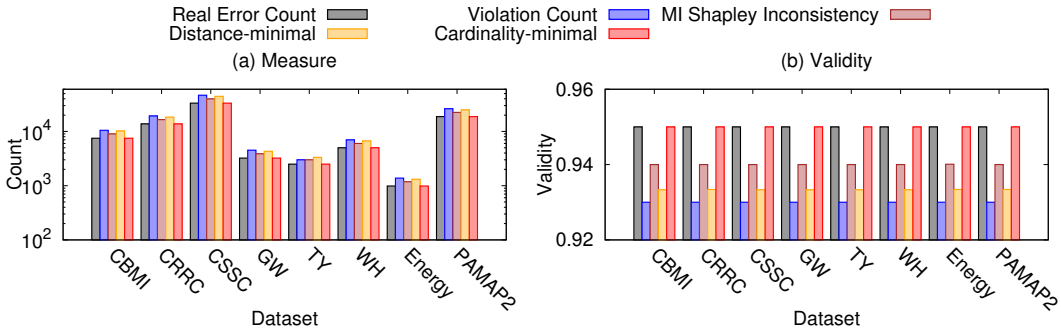


Fig. 19. Inconsistency measures for consecutive errors

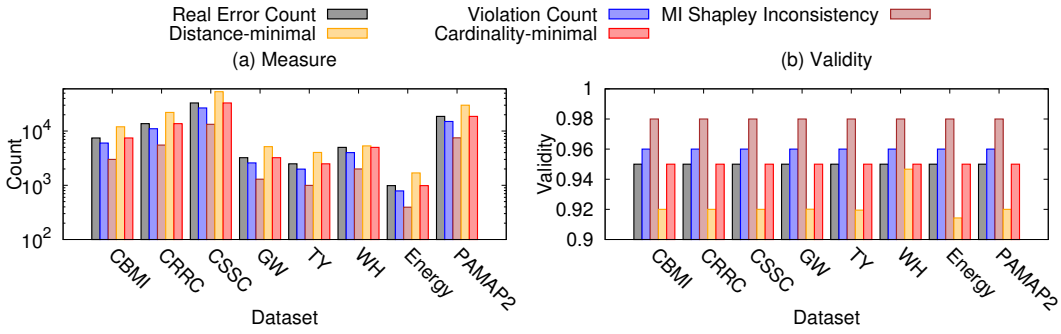


Fig. 20. Inconsistency measures for shift errors

the shifted segment to obtain a time series satisfying both constraints, i.e., violation count under-claiming. As presented in Figure 20, the violation count is smaller than the number of shift errors injected in the data. Such under-claiming of shift error may not be prevalent, e.g., not observed largely in Section 6.7.

6.6.2 Comparison to MI Shapley Inconsistency. The MI Shapley Inconsistency [18] measures the number of minimal inconsistent subsets, i.e. , the number of pairs violating the speed constraints and points violating the value constraints for time series. It may also over-claim or under-claim the

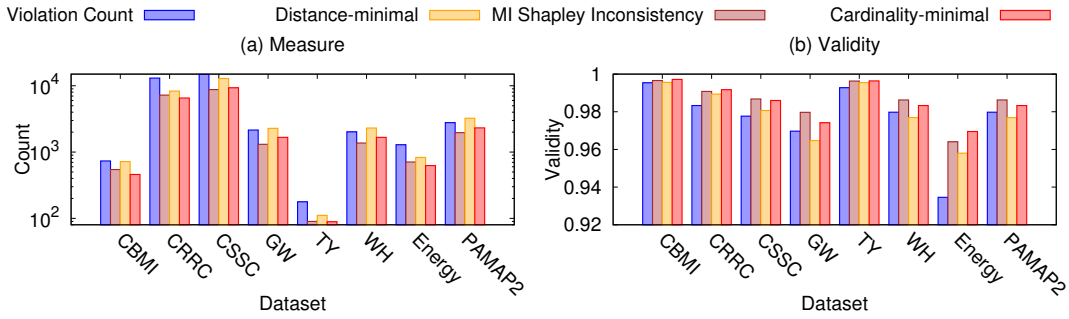


Fig. 21. Validity by various inconsistency measures

data validity issue. As illustrated in Figure 15 of spike errors, four subsets each with two points (denoted by the light blue lines) are counted as minimal inconsistent subsets, while only 2 points need to be changed. Similar over-claiming is also observed in Figure 16 of consecutive errors. Thereby, in Figures 18 and 19, the MI Shapley Inconsistency count is larger than the number of spike and consecutive errors injected.

For shift errors, though a large number of points shift in Figure 17, only two subsets are considered as minimal inconsistent subsets. One subset consists of the start point of shifted segment and its preceding neighbor, and the other contains the end point of shifted segment and its succeeding neighbor. That is, the MI Shapley Inconsistency under-claims shift errors. As shown in Figure 20, its count is smaller than the number of shift errors injected.

6.6.3 Comparison to Distance-minimal Cost. The distance-minimal cost [9] follows the principle that the total difference between the original points and the changed points is minimized. For spike errors in Figure 15, the distance-minimal criteria also needs only 1 point (marked with red cross) to be changed to satisfy the constraints, the same as the cardinality-minimal cost. That is, both are precise for measuring spike errors. Consequently, in Figure 18, they are equal to the number of spike errors injected in the data.

However, the distance-minimal cost cannot always promise the cardinality of changed points is minimal. For example, 9 points with red cross in Figure 17 need to be changed, to minimize the distance cost, whereas only 6 points are shift errors. Similar over-claiming is also observed for consecutive errors in Figure 16. Hence, in Figures 19 and 20, the distance-minimal cost is larger than the number of consecutive and shift errors injected in the data.

6.7 Case Studies

To better demonstrate the usefulness of the validity measure, we present several case studies on how the validity measure is used in our 6 industrial partners and likewise on 2 public datasets.

Figure 21(a) reports the cardinality-minimal cost and other alternative inconsistency measures, while Figure 21(b) compares the validity measures computed by using the aforesaid violation and change costs as $\delta(\mathbf{x})$ in Definition 2. A large number of data points in violation are over claimed with data quality issues, which may not need change. The computed validity is much lower. The large gaps verify the motivation of this study, considering the cardinality-minimal cost for validity.

7 RELATED WORK

Data errors are prevalent and detectable by various methods [6]. Data validity considers the assuring ones, violating the constraints.

7.1 Violation Detection

To evaluate the validity of data, integrity constraints are often employed to detect the violations or inconsistencies [12]. In addition to the widely used functional dependencies [14], Chandel et al. [10] propose to validate collections of user defined constraints on a number of relational tables. These methods for detecting violations in relational databases are not directly applicable to the time series data considered in this study.

Besides the cardinality-minimal cost [7] employed in this paper, the S-repair [17] proposes to delete a minimum number of tuples such that the remaining tuples can satisfy the constraints. Rather than minimizing the deletions, Sun et al. [27] propose to maximize the density of the remaining data. Note that the deletion makes the time series incomplete and it is thus not adopted.

7.2 Outlier Detection

In addition to constraint violation, outliers are also considered in error detection [31]. Outliers are some data points dramatically different than the rest of the data [22]. While violation detection is supervised by constraints, outlier detection over temporal data [13] is often unsupervised. The Distance-based Outlier Detection in Data Streams (DODDS) [29] is to detect data point that has less than K neighbors within a distance of R, for a sliding window.

Autoregressive model is also employed to detect the anomalies in data streams [15, 19]. The outliers could either be errors or abnormalities, such as anomalous behavior of moving objects in trajectory streams [32]. In contrast, the violations to constraints must be errors, e.g., engine torque values should not be negative, and thereby considered in the validity of data.

8 CONCLUSION

To our best knowledge, this is the first study on defining, evaluating, implementing and deploying the data validity measure in a time series database. (1) We identify that simply counting violations of constraints may over claim the time series data quality issue. It motivates us to consider the minimum number of data points that need to be changed in order to satisfy the constraints, or equivalently, the maximum rate of data that can be reserved without change, as the validity measure. (2) We prove in Proposition 4 that there must exist one point having no need to change, unless all the data points violate the value constraints. It leads to efficient algorithms for computing the validity measure in quadratic time and linear space. (3) For a time series stored in multiple segments, we show in Proposition 7 that the validity of two segments could be directly aggregated in certain cases. By re-splitting the time series, more segments are able to directly aggregate on validity. (4) The validity measure has been deployed as a function of Apache IoTDB, an open-source time series database. Two deployments, UDF based and TsFile based, without/with pre-computation, are designed for different scenarios. The experiments show up to 4 orders of magnitude improvement in time performance by our proposal.

ACKNOWLEDGMENTS

This work is supported in part by National Natural Science Foundation of China (62021002, 62072265, 62232005), National Key Research and Development Plan (2021YFB3300500), Beijing National Research Center for Information Science and Technology (BNR2022RC01011), and Alibaba Group through Alibaba Innovative Research (AIR) Program. Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

REFERENCES

[1] https://archive.ics.uci.edu/.
[2] https://github.com/apache/iotdb/tree/research/quality-validity.

[3] https://github.com/iotdbValidity/validity-exp.
[4] https://iotdb.apache.org.
[5] https://iotdb.apache.org/UserGuide/Master/UDF-Library/Data-Quality.html#validity.
[6] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *Proc. VLDB Endow.*, 9(12):993–1004, 2016.
[7] G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints. *Proc. VLDB Endow.*, 3(1):197–207, 2010.
[8] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In C. S. Jensen, C. M. Jermaine, and X. Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 541–552. IEEE Computer Society, 2013.
[9] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In F. Özcan, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 143–154. ACM, 2005.
[10] A. Chandel, N. Koudas, K. Q. Pu, and D. Srivastava. Fast identification of relational constraint violations. In R. Chirkova, A. Dogac, M. T. Özsu, and T. K. Sellis, editors, *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 776–785. IEEE Computer Society, 2007.
[11] Y. Chen and C. Caramanis. Noisy and missing data regression: Distribution-oblivious support recovery. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 383–391. JMLR.org, 2013.
[12] W. Fan, F. Geerts, S. Ma, and H. Müller. Detecting inconsistencies in distributed data. In F. Li, M. M. Moro, S. Ghandeharizadeh, J. R. Haritsa, G. Weikum, M. J. Carey, F. Casati, E. Y. Chang, I. Manolescu, S. Mehrotra, U. Dayal, and V. J. Tsotras, editors, *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 64–75. IEEE Computer Society, 2010.
[13] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. *Outlier Detection for Temporal Data.* Synthesis Lectures on Data Mining and Knowledge Discovery. Morgan & Claypool Publishers, 2014.
[14] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In R. Fagin, editor, *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, volume 361 of *ACM International Conference Proceeding Series*, pages 53–62. ACM, 2009.
[15] N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In L. Cao, C. Zhang, T. Joachims, G. I. Webb, D. D. Margineantu, and G. Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1939–1947. ACM, 2015.
[16] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang. Cleanml: A study for evaluating the impact of data cleaning on ML classification tasks. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 13–24. IEEE, 2021.
[17] E. Livshits, B. Kimelfeld, and S. Roy. Computing optimal repairs for functional dependencies. In J. V. den Bussche and M. Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 225–237. ACM, 2018.
[18] E. Livshits, R. Kochirgan, S. Tsur, I. F. Ilyas, B. Kimelfeld, and S. Roy. Properties of inconsistency measures for databases. In G. Li, Z. Li, S. Idreos, and D. Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1182–1194. ACM, 2021.
[19] I. Melnyk, A. Banerjee, B. L. Matthews, and N. C. Oza. Semi-markov switching vector autoregressive model-based anomaly detection in aviation systems. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1065–1074. ACM, 2016.
[20] P. E. O'Neil, E. Cheng, D. Gawlick, and E. J. O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.
[21] F. Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, 1994.
[22] D. Samariya and J. Ma. A new dimensionality-unbiased score for efficient and effective outlying aspect mining. *Data Sci. Eng.*, 7(2):120–135, 2022.
[23] S. Song, C. Li, and X. Zhang. Turn waste into wealth: On simultaneous clustering and cleaning over dirty data. In L. Cao, C. Zhang, T. Joachims, G. I. Webb, D. D. Margineantu, and G. Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1115–1124. ACM, 2015.
[24] S. Song and A. Zhang. Iot data quality. In M. d'Aquin, S. Dietze, C. Hauff, E. Curry, and P. Cudré-Mauroux, editors, *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 3517–3518. ACM, 2020.

[25] S. Song, A. Zhang, J. Wang, and P. S. Yu. SCREEN: stream data cleaning under speed constraints. In T. K. Sellis, S. B. Davidson, and Z. G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 827–841. ACM, 2015.

[26] S. Sun, S. Ma, J. Song, W. Yue, X. Lin, and T. Ma. Experiments and analyses of anonymization mechanisms for trajectory data publishing. *J. Comput. Sci. Technol.*, 37(5):1026–1048, 2022.

[27] Y. Sun and S. Song. From minimum change to maximum density: On s-repair under integrity constraints. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 1943–1948. IEEE, 2021.

[28] M. Thimm. On the expressivity of inconsistency measures. *Artif. Intell.*, 234:120–151, 2016.

[29] L. V. Tran, M. Mun, and C. Shahabi. Real-time distance-based outlier detection in data streams. *Proc. VLDB Endow.*, 14(2):141–153, 2020.

[30] D. Z. Wang, X. L. Dong, A. D. Sarma, M. J. Franklin, and A. Y. Halevy. Functional dependency generation and applications in pay-as-you-go data integration systems. In *12th International Workshop on the Web and Databases, WebDB 2009, Providence, Rhode Island, USA, June 28, 2009*, 2009.

[31] P. Wang and Y. He. Uni-detect: A unified approach to automated error detection in tables. In P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 811–828. ACM, 2019.

[32] Y. Yu, L. Cao, E. A. Rundensteiner, and Q. Wang. Detecting moving object outliers in massive-scale trajectory streams. In S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 422–431. ACM, 2014.

[33] A. Zhang, S. Song, and J. Wang. Sequential data cleaning: A statistical approach. In F. Özcan, G. Koutrika, and S. Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 909–924. ACM, 2016.

Received July 2022; revised October 2022; accepted November 2022.