On Repairing Timestamps for Regular Interval Time Series

Chenguang Fang
BNRist, Tsinghua University
fcg19@mails.tsinghua.edu.cn

Shaoxu Song
BNRist, Tsinghua University
sxsong@tsinghua.edu.cn

Yinan Mei
BNRist, Tsinghua University
myn18@mails.tsinghua.edu.cn

ABSTRACT

Time series data are often with regular time intervals, e.g., in IoT scenarios sensor data collected with a pre-specified frequency, air quality data regularly recorded by outdoor monitors, and GPS signals periodically received from multiple satellites. However, due to various issues such as transmission latency, device failure, repeated request and so on, timestamps could be dirty and lead to irregular time intervals. Amending the irregular time intervals has obvious benefits, not only improving data quality but also leading to more accurate applications such as frequency-domain analysis and more effective compression in storage. The timestamp repairing problem however is challenging, given many interacting factors to determine, including the time interval, the start timestamp, the series length, as well as the matching between the time series before and after repairing. Our contributions in this paper are (1) formalizing the timestamp repairing problem for regular interval time series to minimize the cost w.r.t. move, insert and delete operations; (2) devising an exact approach with advanced pruning strategies based on lower bounds of repairing; (3) proposing an approximation based on bi-directional dynamic programming. The experimental results demonstrate the superiority of our proposal in both timestamp repair accuracy and the aforesaid applications. Remarkably, the repair results can be used to evaluate time series data quality measures. Both the repair and measure functions have been implemented in an open-source time series database, Apache IoTDB.

PVLDB Reference Format:
Chenguang Fang, Shaoxu Song, and Yinan Mei. On Repairing Timestamps for Regular Interval Time Series. PVLDB, 15(9): 1848 - 1860, 2022.
doi:10.14778/3538598.3538607

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/fangfcg/regular-interval-repair.

1 INTRODUCTION

Large amounts of time series are often with regular time intervals, e.g., in IoT scenarios, sensor data collected periodically [21], air quality data regularly recorded by outdoor monitors [32], and GPS signals periodically received from multiple satellites [9]. The devices are often programmed to collect data in a pre-specified frequency. Such regular interval time series reflects not only the trend of the values, but also makes it easier for analysis and storage.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 15, No. 9 ISSN 2150-8097.
doi:10.14778/3538598.3538607

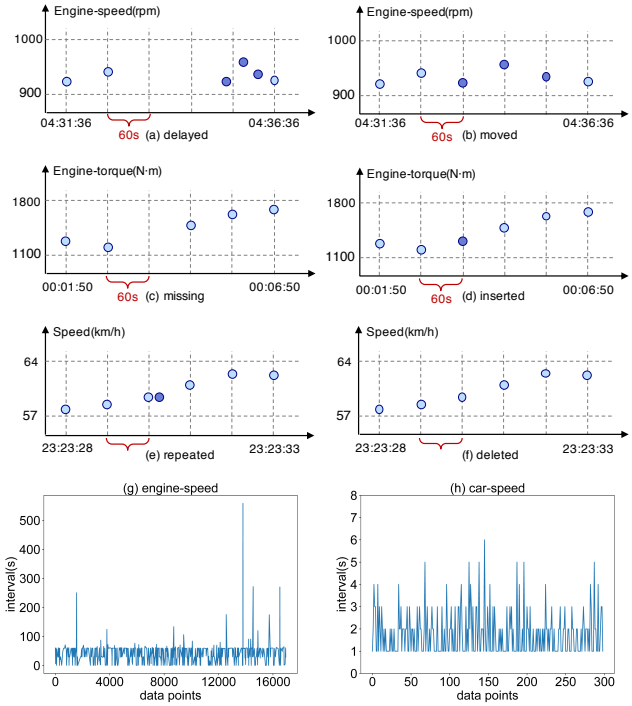

Figure 1: Real vehicle data suffering from irregular time intervals, where dark blue points are (a) delayed / (b) moved, (c) missing / (d) inserted, and (e) repeated / (f) deleted. While points in (g) are largely collected with time intervals about 60s, the most likely time interval of (h) is unclear.

1.1 Irregular Time Interval by Dirty Timestamp

Owing to various issues illustrated below, timestamps could be imprecise, and thereby lead to irregular time intervals in time series.

(1) Transmission latency. IoT data are often transmitted from sensors with no clocks installed, and timestamped till they are received by the terminal (clock enabled). Due to the instability of network latency, often non-constant, as illustrated in Figure 1(a), the data regularly generated in sensors can be timestamped irregularly, thereby leading to irregular intervals with timestamp shifts [22].

(2) Missing observations. Even worse, due to transmission errors [27] or device failures such as broken sensors, the data could also be missing, as shown in Figure 1(b). It is worth noting that most sensors are stateless, and thus using a simple counter as clock is not an option and imprecise due to the missing points.

(3) Repeated requests. Data transmission mechanisms such as Automatic Repeat reQuest (ARQ) [24] may send packets repeatedly over an unreliable communication channel, to achieve reliable data

transmission. However, overly repeated transmissions might result in redundant points in a short period, e.g., in Figure 1(c).

The proposal repairs all the cases including delayed, missing and repeated points, by moving as well as inserting and deleting.

Example 1 (Real Vehicle Scenario)**.** In our partner company, a heavy machine maintenance service provider, each vehicle installs at least 13 free-running sensors, monitoring engine speed, temperature, etc. These sensors do not have their own clocks, instead, sending data directly to the data bus installed in the terminal of the vehicle. The terminal runs Linux with a clock. Each data is then timestamped when the terminal collects it from the bus.

Figure 1 illustrates three examples of irregular interval time series in (a), (c) and (e) from the aforesaid vehicle scenario, and their corresponding repairs in (b), (d) and (f), respectively. Figure 1(g) presents the time intervals of consecutive data points of Engine-speed. While the time intervals could vary significantly, most of them are about 60s. However, owing to the extremely noisy data, the interval of the Speed data in Figure 1(h) is unclear (1s or 2s).

Handling irregular time intervals is meaningful not only in business (OT) but also in data management (IT). Experiments on some case studies in Section 6.3 show that handling irregular time intervals enables FFT analysis and improves storage space.

For example, to predict vehicle fuel consumption, machine learning models are applied, which often take regular interval time series as input, such as LSTM [16, 33]. Likewise, to detect abnormal engine behaviors, e.g., rotational axis deviation, frequency-domain analysis techniques are employed, such as Fast Fourier Transform (FFT) [25], again requiring regular interval time series. The cases of delayed, missing and repeated points lead to irregular time intervals, and thus obscure these learning and analysis models performing.

Moreover, to efficiently store time series data, delta-like encoding and compression techniques are often employed, e.g., in Apache IoTDB [1]. The idea is to store the deltas of consecutive points, often small and repetitive, rather than the original large timestamps and values. The irregular time intervals lead to large and non-repetitive deltas, degrading the encoding and compression performance.

Indeed, detecting these interacting events is highly non-trivial, e.g., whether a point is delayed or repeated. It needs a holistic consideration of possible cases, which does not only detect the events but also outputs the possible repairs as a by-product. The result in a form of $(*, s_j)$ or $(t_i, *)$ or (t_i, s_j) denotes that a point is missing at s_j, or repeated at t_i, or delayed from s_j to t_i, respectively. After handling irregular time intervals to regular ones, value repair is then conducted, which is not the focus of this study. Experiments in Section 6.4 show how this proposal works together with value repair and jointly contributes to the FFT analysis in Section 6.3. ■

1.2 Repairing for Regular Interval Time Series

In this paper, we mainly detect and repair the data quality issues in timestamps, leading to delayed, missing and repeated points. While moving a delayed point changes only timestamp but not its value, deleting a repeated point does not need to handle the value either. However, for inserting a missing point, both its timestamp and value should be repaired. Note that missing value imputation has been extensively studied [19, 23]. Thereby, we focus on repairing the timestamps of the missing points and apply the existing value imputation after timestamp repairing as illustrated in Section 6.4.

The challenge of repairing timestamps for regular interval time series is due to the following multiple interacting factors. (1) **Matching**. The matching between the time series before and after repairing with the minimum distance. (2) **Start**. The start of the time series, which could involve dirty timestamps as well. (3) **Length**. Taking missing and redundant points into consideration, it is difficult to determine the length of the repaired time series. (4) **Interval**. While the data collection frequency may be curated as metadata in some scenarios (e.g., every 7s for Wind Turbine), precisely determining the real time interval could also be difficult especially when the timestamps are extremely noisy, as shown in Figure 1(h).

1.3 Contributions

Our major contributions are summarized as follows.

(1) We formalize the timestamp repairing problem for regular interval time series to minimize the cost of move, insert and delete operations. The determinations of interval, start and length in regular interval time series are considered as sub-problem (Section 2).

(2) We devise an exact approach based on dynamic programming, to find the matching between irregular and regular interval time series with the minimum distance cost, as well as determine the length of regular interval time series. Remarkably, we prove the lower bounds of the repair cost in terms of the interval and the start, therefore developing advanced pruning strategies (Section 3).

(3) An approximation algorithm is devised based on bi-directional dynamic programming, significantly reducing time cost (Section 4).

(4) We conduct comprehensive evaluations against various existing methods, in both repaired timestamps and downstream tasks, e.g., frequency-domain analysis and data compression (Section 6).

Our approach is implemented as a function `timestamprepair` in an open-source time series database, Apache IoTDB [1, 2], together with three data quality measures, including `timeliness`, `completeness` and `consistency`. All the proofs can be found in [3].

2 PROBLEM STATEMENT

In this section, we first introduce the definition of regular interval time series, and then formalize the repair problem.

2.1 Preliminaries

Definition 1 (Regular Interval Time Series)**.** *A regular interval time series is the time series with regular time intervals of consecutive data points. For a regular interval time series* $\mathbf{s}$*, we use* $\{s_0, s_1, \ldots, s_{m-1}\}$ *to denote the timestamps, where* s_i *is the timestamp of the* i*-th data point. Since the intervals of consecutive timestamps are regular, we have* $s_i = s_0 + i \cdot \epsilon_T, i = 0, 1, \ldots, m-1$*, where* s_0 *is the start,* ϵ_T *is the time interval and* m *is the length.*

Consider a time series $\mathbf{t}$ with timestamps $\{t_0, t_1, \ldots, t_{n-1}\}$ of size n, where t_i is the timestamp of the i-th value. Since the intervals (i.e., $\{t_i - t_{i-1} \mid i = 1, 2, \ldots, n-1\}$) of $\mathbf{t}$ might not be regular, we propose to repair the original time series $\mathbf{t}$ into a regular interval time series $\mathbf{s}$, while trying to minimize the change as in data repairing [20].

The length n of the original time series $\mathbf{t}$ is a part of the input $\mathbf{t}$, whereas the length m of the target regular interval time series

$\mathbf{s}$ could either be specified by the input (if known already) as in Problem 2 or determined by Algorithm 1 as in Problem 3.

2.2 Repair Problem

To repair the time series, three repair operations are introduced, including move, insert and delete, corresponding to the data quality issues of delayed points, missing points and repeated points introduced in Figure 1. We use Figure 2(d) to outline the operations.

Definition 2 (Move Cost). *Move cost refers to the cost of modification of the timestamp, i.e., move an original timestamp t_i in $\mathbf{t}$ to another timestamp s_j in $\mathbf{s}$, denoted by $\Delta_m(t_i, s_j) = |t_i - s_j|$.*

Definition 3 (Insert Cost). *Insert cost refers to the cost of inserting new timestamps, i.e., impute missing timestamps. Since the new timestamp will be directly introduced into $\mathbf{s}$, we define insert cost as $\Delta_a(s_j) = \lambda_a$, where λ_a is a given constant.*

Definition 4 (Delete Cost). *Delete cost refers to the cost of deleting timestamps in $\mathbf{t}$, denoted by $\Delta_d(t_i) = \lambda_d$, where λ_d is a given constant.*

We then define a match, to denote the mapping relationship from original sequence $\mathbf{t}$ to the target regular interval sequence $\mathbf{s}$.

Definition 5 (Match). *A match is a set of point pairs from the original time series $\mathbf{t}$ to the target regular interval time series $\mathbf{s}$. Since we allow move, insert and delete operations, a match is defined based on the aforesaid three operations, $M = \{(x, y) \mid x \in \{t_0, t_1, \ldots, t_{n-1}, *\}, y \in \{s_0, s_1, \ldots, s_{m-1}, *\}\}$, where $(t_i, s_j), (t_i, *), (*, s_j)$ denote the move, delete and insert operations, respectively. All the timestamps in $\mathbf{t}$ and $\mathbf{s}$ should be included only once in M, which ensures M a complete mapping relationship from $\mathbf{t}$ to $\mathbf{s}$ without any overlapping.*

Finally, the repair cost of the problem is defined based on the original time series $\mathbf{t}$, target time series $\mathbf{s}$ and the match M.

Definition 6 (Repair Cost). *Given $\mathbf{t}$, $\mathbf{s}$ and a match M, the repair cost comprises move cost, delete cost and insert cost from $\mathbf{t}$ to $\mathbf{s}$.*

$$\begin{aligned}\Delta(\mathbf{t}, \mathbf{s}, M) &= \sum_{(t_i, s_j) \in M} \Delta_m(t_i, s_j) + \sum_{(t_i, *) \in M} \Delta_d(t_i) + \sum_{(*, s_j) \in M} \Delta_a(s_j) \\ &= \sum_{(t_i, s_j) \in M} |t_i - s_j| + \sum_{(t_i, *) \in M} \lambda_d + \sum_{(*, s_j) \in M} \lambda_a\end{aligned}$$

Following the minimum data repairing principle [20], we aim to minimize the repair cost by finding a proper regular interval time series $\mathbf{s}$ as the target for repairing.

Problem 1 (Regular Interval Time Series Repair). *Given an original time series $\mathbf{t}$ of size n, with timestamps $\{t_0, t_1, \ldots, t_{n-1}\}$, the Regular Interval Time Series Repair problem is to find a regular interval time series $\mathbf{s}$ and a match M such that the repair cost $\Delta(\mathbf{t}, \mathbf{s}, M)$ is minimized, i.e., $\mathbf{s}, M = \arg\min_{\mathbf{s}, M} \Delta(\mathbf{t}, \mathbf{s}, M)$*

In the repair problem, given the original time series $\mathbf{t}$ of size n as input, we do not have any assumptions on knowing the positions (e.g., the i-th points) of the missing, delayed and duplicate values.

2.3 Sub-Problems

In this section, by considering sub-problems of Problem 1, the solutions are decoupled, and thus applied to various scenarios. We first consider the most specific scenario, when the interval ϵ_T, the start s_0 and the length m of $\mathbf{s}$ are all acquired. Intuitively, given ϵ_T, s_0 and m, the target $\mathbf{s}$ should have been uniquely determined. There should be a match M for each point from $\mathbf{t}$ to $\mathbf{s}$ that minimizes the repair cost. The Match Searching problem is thus to find such M.

Table 1: Relationship of the sub-problems

Sub-problem	Input factors	Target factors
Match Searching	ϵ_T, s_0, m	M
Length Determination	ϵ_T, s_0	m, and also M
Start Determination	ϵ_T	s_0, and also m, M
Interval Determination	–	ϵ_T, and also s_0, m, M

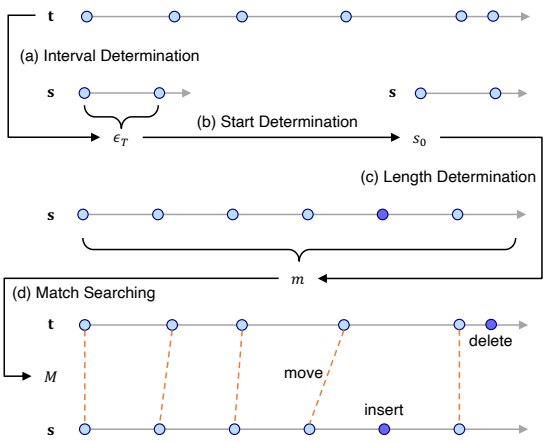

Figure 2: The sub-problems and pipeline from the input time series $\mathbf{t}$ to the regular interval time series $\mathbf{s}$.

Problem 2 (Match M Searching). *Given a time series $\mathbf{t}$ and a regular interval time series $\mathbf{s}$, the Match Searching problem is to find a match M that minimizes the repair cost $\Delta(\mathbf{t}, \mathbf{s}, M)$, i.e.,*

$$\begin{aligned}M &= \arg\min_{M} \Delta(\mathbf{t}, \mathbf{s}, M) \\ &= \arg\min_{M} \sum_{(t_i, s_j) \in M} |t_i - s_j| + \sum_{(t_i, *) \in M} \lambda_d + \sum_{(*, s_j) \in M} \lambda_a\end{aligned}$$

Next, we generalize the scenario when the interval ϵ_T and start s_0 are known. The problem is thus to determine the length m of $\mathbf{s}$.

Problem 3 (Length m Determination). *Given an original time series $\mathbf{t}$, the interval ϵ_T and the start s_0 of the target time series $\mathbf{s}$, the Length Determination problem is to find the best length m^* of $\mathbf{s}$, such that the repair cost $\Delta(\mathbf{t}, \mathbf{s}, M)$ can be minimized.*

We further generalize the problem by giving only the interval ϵ_T, while both the length m and the start s_0 are unknown. The Start Determination problem is thus to find the optimal start for $\mathbf{s}$, which can be solved by finding the optimal length m for each possible start s_0, calling the solutions of Problem 3 Length Determination.

Problem 4 (Start s_0 Determination). *Given an original time series $\mathbf{t}$ and the interval ϵ_T of the target time series $\mathbf{s}$, the Start Determination*

problem is to find the best start s_0^* of $\mathbf{s}$*, such that the repair cost* $\Delta(\mathbf{t}, \mathbf{s}, M)$ *can be minimized.*

Finally, we consider the general scenario when the interval ϵ_T is not given either (owing to the reasons in Section 1). The Interval Determination is to find the optimal interval for $\mathbf{s}$, and again can be solved by finding the best start s_0 for all possible intervals ϵ_T, calling the solutions of Problem 4 Start Determination.

Problem 5 (Interval ϵ_T Determination)**.** *Given an original time series* $\mathbf{t}$*, the Interval Determination problem is to find the interval* ϵ_T^* *of* $\mathbf{s}$*, such that the repair cost* $\Delta(\mathbf{t}, \mathbf{s}, M)$ *can be minimized.*

Figure 2 illustrates the order of determining the parameters, while Table 1 highlights the input and output of the sub-problems. Given an input time series $\mathbf{t}$, to repair it to a regular interval time series $\mathbf{s}$, Interval Determination (Problem 5) first computes the interval ϵ_T of $\mathbf{s}$. Next, Start Determination (Problem 4) determines its start s_0. Then, Length Determination (Problem 3) finds the optimal length m for $\mathbf{s}$ according to the interval and start. Finally, Match Searching (Problem 2) find the match M from $\mathbf{t}$ to $\mathbf{s}$.

The reason for determining the match from $\mathbf{t}$ to $\mathbf{s}$ at the bottom is that it replies on a known $\mathbf{s}$ with ϵ_T, s_0, m determined. Since the optimal length m could be found together with the match as shown in Algorithm 1, Length Determination is thus the step before Match Searching. For ϵ_T and s_0, we choose to determine the interval ϵ_T first, since the start s_0 could be efficiently pruned by the interval according to Corollary 7.

3 EXACT METHOD

In this section, according to the different scenarios and sub-problems defined in Section 2, we devise corresponding solutions.

3.1 Match Searching

3.1.1 Match Searching Algorithm. Referring to Definitions 5 and 6 of match and repair cost, we devise a polynomial-time algorithm based on dynamic programming for the match searching problem.

Let $\mathbf{t}_{[i]}$ denote the subsequence $\{t_0, t_1, \ldots, t_{i-1}\}$ of sequence $\mathbf{t}$, i.e., the first i items in $\mathbf{t}$, and analogously, let $\mathbf{s}_{[j]}$ denote the subsequence $\{s_0, s_1, \ldots, s_{j-1}\}$ of sequence $\mathbf{s}$. We will introduce the state transition equation for the dynamic programming algorithm.

Let $dp(i, j)$ denote the minimum repair cost from time series $\mathbf{t}_{[i]}$ to $\mathbf{s}_{[j]}$, we have the following state transition equation

$$dp(i,j) \leftarrow \min(dp(i-1,j-1) + \Delta_m(t_{i-1}, s_{j-1}), \quad dp(i,j-1) + \Delta_a(s_{j-1}), \quad dp(i-1,j) + \Delta_d(t_{i-1})) \tag{1}$$

where Δ_m, Δ_a and Δ_d are move, insert and delete costs, respectively. The state transition equation minimizes the cost among the three possible cases of move, insert and delete operations in matching.

Proposition 1. *The state transition equation* $dp(i, j)$ *in Formula 1 gives the minimum repair cost from subseries* $\mathbf{t}_{[i]}$ *to* $\mathbf{s}_{[j]}$*.*

Algorithm 1 shows the procedure of Match Searching. Dynamic programming (Lines 8-12) is conducted according to Formula 1. After finding the minimum cost, we then traceback to find the match M by Algorithm 2.

Algorithm 1: match searching $(\mathbf{t}, \epsilon_T, s_0)$

Input: original time series $\mathbf{t}$, time interval ϵ_T, start s_0
Output: the match M from $\mathbf{t}$ to $\mathbf{s}$, the length m^* of $\mathbf{s}$

1 $n \leftarrow Length(\mathbf{t})$;
2 init $DP[n][*], OP[n][*], V[n][*]$ // record dp results, operations & value state;
3 $DP[0][0] \leftarrow 0, DP[0][p] \leftarrow p\lambda_a, DP[q][0] \leftarrow q\lambda_d$;
4 $m^*, m_{ub}, min\_cost \leftarrow +\infty$;
5 $m \leftarrow 1$;
6 **while** $m \leq m_{ub}$ **do**
7 **for** $i \leftarrow 1$ *to* n **do**
8 Update $DP[i][m]$ according to Formula 1;
9 **if** $|minimum\_operations| \geq 2$ **then**
10 choose *operation* with the largest $V[i][m]$;
11 Update $V[i][m]$ according to Formulas 2-4;
12 $OP[i][m] \leftarrow 0/1/2$ //0: move, 1: insert, 2: delete ;
13 **if** $DP[n][m] < min\_cost$ **then**
14 $min\_cost, m^* \leftarrow DP[n][m], m$;
15 $m_{ub} \leftarrow \lfloor \frac{min\_cost}{\lambda_a} + n \rfloor$;
16 $m \leftarrow m + 1$
17 $M \leftarrow \text{traceback}(OP[n][m^*], \mathbf{t}, \mathbf{s}_{[m^*]})$;
18 **return** M, m^*;

3.1.2 Using Data Characteristics. Note that Algorithm 1 may find more than one match with the minimum repair costs. Intuitively, the characteristics of the data values may help in making a further decision. Figure 3(a) gives an example of searching for the match M. Given $s_j - t_i = t_{i+1} - s_j = \sigma$, deleting any of them and moving the other to s_j have the same repair costs. However, in Figure 3(b), $v_{i-1} = v_i$ indicates that v_i is probably the repeated point and needs to be deleted, where v_{i-1} and v_i are the data values of $\mathbf{t}$ at t_{i-1} and t_i, respectively.

Formally, let $\mathbf{v} = \{v_0, v_1, \cdots, v_{n-1}\}$ be the values for the input time series $\mathbf{t}$, where v_i is the corresponding data value at timestamp t_i. To employ data values for deciding match, we first construct the value state matrix V of size $n \times m$, updating together with the dynamic state dp in Formula 1. By considering the three data quality issues in the value domain, the updating of V is introduced as follows:

(a) For the delayed point, let (t_i, s_j) denote the corresponding move operation, and $\delta_{v_i} = |v_i - v_{i-1}|$ be the value fluctuation at t_i. Intuitively, a delayed t_i should affect both the value fluctuations before and after it. That is, the value fluctuations δ_{v_i} and $\delta_{v_{i+1}}$ before and after t_i are supposed to be close. We thus update $V(i, j)$ as

$$V(i,j) = V(i-1,j-1) + \frac{\min(\delta_{v_i}, \delta_{v_{i+1}})}{\max(\delta_{v_i}, \delta_{v_{i+1}})}. \tag{2}$$

(b) For the repeated point, the data values are probably the same as the values before them. Let $(t_i, *)$ denote the corresponding delete operation, we update $V(i, j)$ as

$$V(i,j) = \begin{cases} V(i,j-1) + 1, & \delta_{v_i} = 0 \\ V(i,j-1), & \delta_{v_i} \neq 0 \end{cases} \tag{3}$$

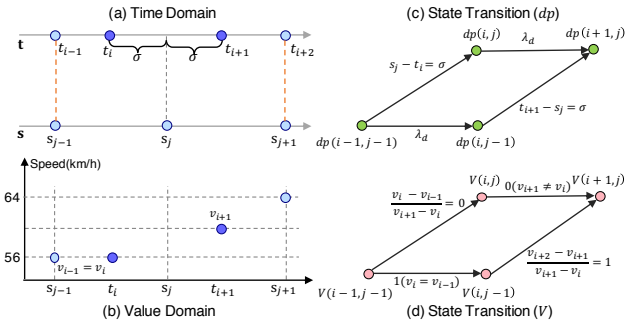

Figure 3: Example of using values for deciding the match

(c) For the missing point, let $(*, s_j)$ denote the corresponding insert operation, and $t^l_{s_j}, t^r_{s_j}$ be the closest timestamps before and after s_j. Since s_j is missing in $\mathbf{t}$, $t^l_{s_j}$ and $t^r_{s_j}$ skipping one point probably have a larger value fluctuation. We thus update $V(i,j)$ as

$$V(i,j) = \begin{cases} V(i,j-1)+1, & |v^l_{s_j} - v^r_{s_j}| \geq \delta_v^{avg} \\ V(i,j-1), & |v^l_{s_j} - v^r_{s_j}| < \delta_v^{avg} \end{cases} . \quad (4)$$

where δ_v^{avg} is the average of $\delta_{v_1}, \cdots, \delta_{v_{n-1}}$.

If the match searching algorithm finds more than one choice of the minimum costs, the one with larger V will be selected, and then V will also be updated (Lines 9-11 in Algorithm 1). An example is provided below to illustrate the process.

Example 2. Figures 3(c) and 3(d) give an example of updating $dp(i,j)$ and $V(i,j)$, corresponding to the timestamps and values in Figures 3(a) and 3(b). For $dp(i+1,j)$, the algorithm computes $dp(i,j) + \lambda_d = dp(i,j-1) + \sigma = dp(i-1,j-1) + \sigma + \lambda_d$, i.e., they have the same repair costs. It thus computes $V(i+1,j)$ updated from $V(i,j)$ and $V(i,j-1)$, respectively. Figure 3(d) shows that $V(i,j) + 0 = V(i-1,j-1) + 0 < V(i,j-1) + 1 = V(i-1,j-1) + 2$. $V(i+1,j)$ is thus chosen to be updated from $V(i,j-1)$, i.e., t_i is deleted and t_{i+1} is moved to s_j. ■

3.1.3 Complexity Analysis. For each possible length m, our match searching (Algorithm 1) updates n elements in the state matrix DP. Referring to Lemma 2, the upper bound of m is depended on min_cost. Since the worst case of the cost is to delete all the points, we have $min\_cost \leq n\lambda_d$. In practice, we usually set $\lambda_d = \lambda_a$, i.e., the same cost for insertion and deletion, as discussed in the parameter selection in Section 3.4.4. According to Lemma 2, we have $m \leq (\frac{\lambda_d}{\lambda_a} + 1)n = 2n$. Traceback (Algorithm 2) employed in Algorithm 1 is conducted $(n + m^*)$ times to find match M. Since $m^* \leq m_{ub} = 2n$, Algorithm 2 thus runs in $O(n)$ time. Therefore, Algorithm 1 runs in $O(n^2)$ time.

3.2 Length Determination

After proposing the solution for Match Searching problem, we move to the determination of length m for $\mathbf{s}$, when the interval ϵ_T and the start s_0 are both known, i.e., Problem 3.

To enable pruning, we propose an upper bound of m^*, based on an arbitrary computed cost.

Algorithm 2: traceback $(OP[n][m], \mathbf{t}, \mathbf{s})$

Input: operation matrix $OP[n][m]$, original time series $\mathbf{t}$, target regular interval time series $\mathbf{s}$
Output: the match M recorded by the operation matrix

```
1  n, m ← Length(t), Length(s);
2  i ← n − 1, j ← m − 1 ;
3  M ← ∅;
4  while i ≥ 0 and j ≥ 0 do
5      if OP[i][j] == 0 then
6          M ← {(t_i, s_j)} ∪ M;
7          i ← i − 1, j ← j − 1;
8      else if OP[i][j] == 1 then
9          M ← {(∗, s_j)} ∪ M;
10         j ← j − 1;
11     else
12         M ← {(t_i, ∗)} ∪ M;
13         i ← i − 1;
14 return M;
```

Lemma 2. *Suppose that we have a repair for* $\mathbf{t}$ *to a regular interval time series* $\mathbf{s}$ *with repair cost* c*, the upper bound of* m *is computed as:*

$$m^* \leq \lfloor \frac{c}{\lambda_a} + n \rfloor \quad (5)$$

To find the optimal length m, the method needs to enumerate all possible m values under the bound proposed in Lemma 2. Indeed, to make the enumeration practical, we update the repair cost c as well as the corresponding bound in Lemma 2, when a smaller cost is obtained, in Line 15 in Algorithm 1. That is, the bound dynamically decreases during the process of enumerating m.

3.3 Start Determination

As discussed before Problem 4, to find the best start s_0^* given the interval ϵ_T, we can call the solution in Section 3.2, i.e., Algorithm 1, to find the optimal length m and match M for each possible start s_0 and return the one with the minimum repair cost.

In this subsection, we first give a simple bound for s_0^*, and further restrict it in Section 3.4, by combining the overall bound for cost.

Lemma 3. *Given the original time series* $\mathbf{t}$ *of length* n *and the interval* ϵ_T *of the regular interval time series* $\mathbf{s}$*, the start* s_0^* *of* $\mathbf{s}$ *satisfies* $t_0 - (\lambda_a + \lambda_d) \leq s_0^* \leq t_{n-1}$.

3.4 Interval Determination

When all the factors are unknown, as introduced in Problem 5, we need to determine the ϵ_T first. In this section, we propose a lower bound of cost w.r.t. the interval, to reduce the time cost of determining ϵ_T by pruning.

3.4.1 Bound for the Cost. We introduce two properties of the minimum match, based on which a bound for the cost is derived.

Lemma 4 (Properties of the Minimum Cost)**.** *Given the original time series* $\mathbf{t}$ *and target regular interval time series* $\mathbf{s}$*, let the match* M *from* $\mathbf{t}$ *to* $\mathbf{s}$ *generate the minimum cost, and* $\lambda_a, \lambda_d \geq \epsilon_T$*. Consider a subsequence of* M *that includes all the move operations in time order,*

denoted as $M^m = \{(t_0^m, s_0^m), (t_1^m, s_1^m), \dots (t_p^m, s_p^m)\} \subseteq M$ *of size* p*, we have the following properties for* M*:*

(1) For any adjacent move operations $(t_i^m, s_i^m), (t_{i+1}^m, s_{i+1}^m) \in M^m$*, there does not exist an insert operation* $(*, s_b)$ *and a delete operation* $(t_a, *)$ *that both in this period of time, i.e.,* $\nexists (*, s_b), (t_a, *) \in M, (t_i^m < t_a < t_{i+1}^m) \wedge (s_i^m < s_b < s_{i+1}^m)$.

(2) There is no insert operation before (t_0^m, s_0^m) *or after* (t_p^m, s_p^m)*, i.e.,* $\forall (*, s_b) \in M, s_0^m < s_b < s_p^m$ *and* $\forall (t_a, *) \in M, t_0^m < t_a < t_p^m$.

Proof sketch. (1) Let insert and delete operations $(*, s_b), (t_a, *) \in M$ be in the same period between $(t_i^m, s_i^m), (t_{i+1}^m, s_{i+1}^m) \in M^m$. Without loss of generality, we assume $s_b < t_a$ (otherwise, we can reverse the sequences). Then we consider two scenarios:

(1.a) If $t_a < s_{i+1}^m$, since $s_b < t_a < s_{i+1}^m$ and $\mathbf{s}$ is the regular interval time series, there must exist $s_b \le s_x < s_{i+1}^m$ that $|s_x - t_a| < \epsilon_T$ and s_x is a delete point. Therefore, we can safely replace $(*, s_x), (t_a, *)$ with (t_a, s_x) which generates lower cost, since $|s_x - t_a| < \epsilon_T < \lambda_a + \lambda_d$, and it contradicts the minimal cost assumption.

(1.b) If $t_a \ge s_{i+1}^m$, we therefore get $s_{i+1}^m \le t_a < t_{i+1}^m$, i.e., $|t_a - s_{i+1}^m| < |t_{i+1}^m - s_{i+1}^m|$. That is, we can replace (t_{i+1}^m, s_{i+1}^m) with (t_a, s_{i+1}^m) for lower cost, which also contradicts the assumption.

In summary, we prove that the insert and delete operations could not appear simultaneously in adjacent move operations.

(2) If k insert operations are at the start of the sequence, i.e, $(*, s_0), (*, s_1), \dots, (*, s_{k-1}) \in M$, we can directly remove these insert operations from M and move the start s_0 to s_k, thus reduce the cost by $k\lambda_a$, which contradicts the minimal cost assumption. ■

In short, the first condition emphasizes that the insert and delete operations could not appear simultaneously in adjacent move operations, and the second condition ensures insert operations could not appear at the start or the end of the sequence.

Proposition 5 (Overall Bound)**.** *Given a time series* $\mathbf{t}$ *and an interval* ϵ_T *of the target regular interval time series, for any* $\mathbf{s}$ *with interval* ϵ_T *and corresponding match* M *from* $\mathbf{t}$ *to* $\mathbf{s}$*, if* $\lambda_d \ge \epsilon_T$ *and* $\lambda_a \ge \epsilon_T$*, we have the bound for* $\Delta(\mathbf{t}, \mathbf{s}, M)$ *as follows:*

$$\Delta(\mathbf{t}, \mathbf{s}, M) \ge \frac{\sum_{i=1}^{n-1} \min\left(|\epsilon_T - \epsilon_i|, \lambda_d\right)}{2}$$

where $\epsilon_i = t_i - t_{i-1}$.

Corollary 6 (Monotonicity of the Overall Bound)**.** *If* $|\epsilon_T - \epsilon_i| \le \lambda_d, i = 1, 2, \dots, n-1$*, the bound* $\frac{\sum_{i=1}^{n-1} \min\left(|\epsilon_T - \epsilon_i|, \lambda_d\right)}{2}$ *is monotonically increasing when* $\epsilon_T \ge \epsilon_{md}$*, and monotonically decreasing when* $\epsilon_T \le \epsilon_{md}$*, where* $\epsilon_{md} = \text{median}(\epsilon_1, \epsilon_2, \dots, \epsilon_{n-1})$.

To find ϵ_T efficiently, we will start the traverse from ϵ_{md} and increase or decrease it gradually, not exceeding the given bound. Analogous to the dynamic bound for m in Section 3.2, the bound for ϵ_T is also dynamically updated, based on the minimum cost.

3.4.2 Pruning Start by the Bound . Recall that we give a bound for start in Lemma 3, which is loose to some extent, making it difficult for traversal. After showing the bound for overall cost, the intuition is to use the overall bound for pruning the traversal of the start.

Corollary 7 (Start Lower Bound)**.** *Given the original time series* $\mathbf{t}$*, and the interval* ϵ_T *of the target regular interval time series* $\mathbf{s}$*, if the first* d *points* $t_0, t_1, \dots, t_d$ *are deleted, i.e.,* $(t_i, *) \in M, i = 0, 1, \dots, d$*, we have the following bound for the cost:*

$$\Delta(\mathbf{t}, \mathbf{s}, M) \ge d\lambda_d + \frac{\sum_{i=d+1}^{n-1} \min\left(|\epsilon_T - \epsilon_i|, \lambda_d\right)}{2}$$

Algorithm 3: exact regular interval repair RIR-Exact

Input: original time series $\mathbf{t}$
Output: the interval ϵ_T^*, the start s_0^*, the length m^* of the target sequence, the match M

```
1  ε_list ← {ε_1, ε_2, ..., ε_n} ;
2  ε_T ← median(ε_list);
3  Range(ε_T) ← [−∞, +∞];
4  min_cost ← +∞;
5  while Range(ε_T) is not empty do
6      d ← 0;
7      while dλ_d + (Σ_{i=d+1}^{n−1} min(|ε_T − ε_i|, λ_d)) / 2 < min_cost do
8          for t_d − λ_d ≤ s_0 ≤ t_d + λ_d do
9              M, m, cost ← match searching(t, ε_T, s_0) ;
10             if cost < min_cost* then
11                 m*, min_cost, s_0*, ε_T* ← m, cost, s_0, ε_T;
12                 Update Range(ε_T) referring to Proposition 5;
13         d ← d + 1;
14     Find another ε_T ∈ Range(ε_T);
15 return ε_T*, s_0*, m*, M;
```

Corollary 8 (Monotonicity of the Start Lower Bound)**.** *Given the original time series* $\mathbf{t}$*, and the interval* ϵ_T *of the target regular interval time series* $\mathbf{s}$*, if the first* d *points* $t_0, t_1, \dots, t_d$ *are deleted, the lower bound of* $\Delta(\mathbf{t}, \mathbf{s}, M)$ *is monotonically increasing w.r.t.* d.

We therefore show the bound of delete points at the start of the sequence. Given d, s_0^* could be found around the true start in $\mathbf{t}$. The overall algorithm is presented in Algorithm 3.

3.4.3 Complexity Analysis. In the worst case, Algorithm 3 searches all the possible ϵ_T less than λ_a and λ_d. For each ϵ_T, since $min\_cost \le n\lambda_d$, and Line 7 ensures $d\lambda_d + \frac{\sum_{i=d+1}^{n-1} \min\left(|\epsilon_T - \epsilon_i|, \lambda_d\right)}{2} < min\_cost \le n\lambda_d$, it repeats n times at most. Moreover, Line 8 repeats $2\lambda_d$ times. Combining Lines 7-8, for each ϵ_T, it calls Algorithm 1 at most $2n\lambda_d$ times. That is, Algorithm 3 runs in $O(n^3\lambda_d^2)$ time.

3.4.4 Determination of Parameters. To determine λ_a and λ_d, firstly, since handling either missing or repeated points will change the number of data points, an intuition is to set $\lambda_a = \lambda_d$. Otherwise, the algorithm might be biased to delete or insert points. For simplicity, we will use λ to represent λ_a and λ_d in the following. With a too small λ, the algorithm might excessively detect missing and repeated points, since the costs of deletion and insertion are lower than moving. On the other hand, with a too large λ, it makes the deletion and insertion difficult. In this sense, $\lambda = \epsilon_{max} = \max(\epsilon_1, \epsilon_2, \dots, \epsilon_{n-1})$, the maximum interval observed in the time series could be a choice, making the costs of delete and insert operations larger than moving the most distant consecutive points. In addition, Corollary 6 shows that if $|\epsilon_T - \epsilon_i| \le \lambda, i = 1, 2, \dots, n-1$, the bound $\frac{\sum_{i=1}^{n-1} \min\left(|\epsilon_T - \epsilon_i|, \lambda\right)}{2}$ is monotonically increasing when

$\epsilon_T \geq \epsilon_{md}$, and monotonically decreasing when $\epsilon_T \leq \epsilon_{md}$. Since $\epsilon_T, \epsilon_i \leq \epsilon_{max}$, by setting $\lambda = \epsilon_{max}$, we could derive $|\epsilon_T - \epsilon_i| \leq \lambda, i = 1, 2, \ldots, n-1$, and the monotonicity applies. Consequently, Algorithm 3 could efficiently find the maximum and minimum values of ϵ_T according to the overall bound in Proposition 5. To this end, we suggest $\lambda_a = \lambda_d = \lambda = \epsilon_{max}$ in practice.

4 APPROXIMATION METHOD

Motivated by the robustness of the median in handling noisy time series data [29], we make two approximations to improve the efficiency while trying to keep the accuracy.

4.1 Approximate Match Searching

Rather than costly enumerating each possible start as in Section 3.3, we propose median approximation, directly regarding the median timestamp of the original time series $\mathbf{t}$ as the median timestamp of the target $\mathbf{s}$, inspired by the robustness of median. As present below, with this median approximation, we no longer need to traverse all the possible starts for $\mathbf{s}$.

4.1.1 Preliminaries. First, let us define the median point $t_{md} = \text{median}\{t_0, t_1, \ldots, t_m\}$ for $\mathbf{t}$. Let $\mathbf{t}^L_{[i]}, \mathbf{t}^R_{[i]}$ denote the subsequence of length i on the left and right side of t_{md}, i.e., before and after t_{md}, respectively. Note that here we slightly abuse the term subsequence for simplicity, since $\mathbf{t}^L_{[i]}$ actually starts from the left point of t_{md} to t_0, i.e., reversed from the subsequence of $\mathbf{t}$.

Let t_{md} and s_{md} denote the median point of $\mathbf{t}$ and $\mathbf{s}$ respectively, according to the idea of median approximation, we assume $t_{md} = s_{md}$ for approximation. Analogously, let $\mathbf{s}^L_{[j]}, \mathbf{s}^R_{[j]}$ denote the subsequence of $\mathbf{s}$ on the left and right of s_{md} respectively with size j. We will start with the state transition equation for the bi-directional dynamic programming algorithm.

4.1.2 Bi-directional Dynamic Programming. We modify the dynamic programming algorithm proposed in Algorithm 1 to bi-directional dynamic programming for median approximation. Let $dp^L(i, j)$ denote the minimum repair cost from time series $\mathbf{t}^L_{[i]}$ to $\mathbf{s}^L_{[j]}$, and $dp^R(i, j)$ denote the minimum repair cost from time series $\mathbf{t}^R_{[i]}$ to $\mathbf{s}^R_{[j]}$, we have the following state transition equation

$$dp^L(i,j) \leftarrow \min(dp^L(i-1,j-1) + \Delta_m(t_i, s_j), \quad (6)$$
$$dp^L(i,j-1) + \Delta_a(s_j), dp^L(i-1,j) + \Delta_d(t_i))$$
$$dp^R(i,j) \leftarrow \min(dp^R(i-1,j-1) + \Delta_m(t_i, s_j), \quad (7)$$
$$dp^R(i,j-1) + \Delta_a(s_j), dp^R(i-1,j) + \Delta_d(t_i))$$

where Δ_m, Δ_a and Δ_d are the move, insert and delete costs, respectively. The overall state transition equation is the combination of the states in both directions.

$$dp^{BD}(i,j) \leftarrow dp^L(i,j) + dp^R(i,j) \quad (8)$$

Indeed, the state transition equation for the bi-directional dynamic programming algorithm is to conduct the dynamic programming algorithm in both directions. We show the correctness below.

Algorithm 4: approximate regular interval repair RIR-Appr

Input: original time series $\mathbf{t}$
Output: the match M from $\mathbf{t}$ to $\mathbf{s}$, the length m of $\mathbf{s}$

$n \leftarrow Length(\mathbf{t}), n_{md} \leftarrow \lfloor \frac{n}{2} \rfloor, s_{md} \leftarrow \text{median}(\mathbf{t})$;
$\epsilon'_T \leftarrow \text{median}(\{t_1 - t_0, t_2 - t_1, \ldots, t_{n-1} - t_{n-2}\})$;
$m^*, m_{ub}, min\_cost \leftarrow +\infty, m \leftarrow 1$;
initialize DP^L, DP^R, OP^L, OP^R ;
while $m \leq m_{ub}$ **do**
  for $i \leftarrow 1$ *to* n_{md} **do**
    Compute s^L_m, s^R_m ;
    Update $DP^L[i][m], OP^L[i][m]$ (Formula 6);
    Update $DP^R[i][m], OP^R[i][m]$ (Formula 7);
    $DP^{BD}[i][m] \leftarrow DP^L[i][m] + DP^R[i][m]$;
  if $DP^{BD}[n_{md}][m] < min\_cost$ **then**
    $min\_cost, m^* \leftarrow DP^{BD}[n_{md}][m], m$;
    Update m_{ub} (Lemma 2);
  $m \leftarrow m + 1$
$M^L \leftarrow Traceback(OP^L[n_{md}][m^*], \mathbf{t}^{\mathbf{L}}, \mathbf{s}^{\mathbf{L}}_{[m^*]})$;
$M^R \leftarrow Traceback(OP^L[n_{md}][m^*], \mathbf{t}^{\mathbf{R}}, \mathbf{s}^{\mathbf{R}}_{[m^*]})$;
$M \leftarrow M^L \cup \{(t_{md}, s_{md})\} \cup M^R / M^L \cup M^R // n$ is odd / even;
return M, m^*;

Proposition 9 (Correctness of the Bi-directional State Transition)**.** *The state transition equation $dp^{BD}(i, j)$ (Formula 8) gives the minimum repair cost from time series $(\mathbf{t}^L_{[i]}, \mathbf{t}^R_{[i]})$ to $(\mathbf{s}^L_{[j]}, \mathbf{s}^R_{[j]})$.*

It shows the correctness of the dynamic programming, i.e., the recurrence state transition equation $dp^{BD}(i, j)$ of Formula 8 always calculates the minimum repair cost in each step. This ensures the final result of the dynamic programming is the minimum repair cost from $\mathbf{t}$ to $\mathbf{s}^{appr}$ given by the approximate algorithm. It is necessary to guarantee the approximation bound in Proposition 11 as well as the special case of the minimum repair cost in Corollary 12.

Analogously, $dp^L(0, p)$ and $dp^R(0, p)$ are initialized with $p\lambda_a$ for $p \in \{1, 2, \ldots\}$, and $dp^L(q, 0)$ and $dp^R(q, 0)$ are initialized with $q\lambda_d$ for $q \in \{1, 2, \ldots\}$. For simplicity, in the median approximation algorithm, we use m to denote the length of $\mathbf{s}^L_{[m]}$ and $\mathbf{s}^R_{[m]}$. Since the repair cost does not change, Lemma 2 still holds in the new algorithm, which bounds m^* with current minimum repair cost by Formula 5. For an odd n, we have $m^* < \frac{\lfloor \frac{c}{\lambda_a} + n \rfloor - 1}{2}$, and for an even n, we have $m^* < \frac{\lfloor \frac{c}{\lambda_a} + n \rfloor}{2}$. Algorithm 4 shows the overall pipeline of the median approximation algorithm.

4.2 Approximate Interval Determination

Though efficient pruning by various bounds are proposed in Section 3.4, it is still very costly to consider a large number of possible intervals to find the optimal ϵ^*_T for $\mathbf{s}$ in Problem 5. Intuitively, as observed in Figure 1(g), most of the intervals in the original time series $\mathbf{t}$ are close to ϵ^*_T, i.e., about 60s.

Therefore, we propose to compute an approximation ϵ'_T of ϵ^*_T by the median of the intervals in $\mathbf{T}$, i.e.,

$$\epsilon'_T = \epsilon_{md} = \text{median}(\epsilon_1, \epsilon_2, \ldots, \epsilon_{n-1}) \quad (9)$$

where $\epsilon_i = t_i - t_{i-1}$. Line 2 in Algorithm 4 casts the approximate ϵ'_T, following Formula 9.

The median approximation (Algorithm 4) does not enumerate the interval and the start point. The **while** loop runs at most m_{ub} times, which is bounded by $2n$. For each $m \leq m_{ub}$, the matrices DP^L and DP^R are updated n times in total. Therefore, Algorithm 4 runs in $O(n^2)$ time.

4.3 Approximation Bound

To prove the error bound for the approximate algorithm, we first prove the triangle inequality over our proposed repair cost, which restricts the difference between the exact and approximate costs (Lemma 10). Combining with the discussions of the repaired time series, we finally derive an error bound for the approximate algorithm to the optimal solution (Proposition 11).

First, by extending the cost function Δ, we use $\Delta(\mathbf{t}^a, \mathbf{t}^b)$ to denote the minimum repair cost between $\mathbf{t}^a$ and $\mathbf{t}^b$, i.e., $\Delta(\mathbf{t}^a, \mathbf{t}^b) = \min_M \Delta(\mathbf{t}^a, \mathbf{t}^b, M)$, where M minimizing the cost, e.g., computed by Algorithm 1. We present the triangle inequality of the repair cost.

Lemma 10. *For any three time series of arbitrary length $\mathbf{t}^a, \mathbf{t}^b, \mathbf{t}^c$, it always has $\Delta(\mathbf{t}^a, \mathbf{t}^b) \leq \Delta(\mathbf{t}^a, \mathbf{t}^c) + \Delta(\mathbf{t}^c, \mathbf{t}^b)$.*

Let $\mathbf{t}$ denote the input time series, $\mathbf{s}^{exact}$ and $\mathbf{s}^{appr}$ denote the regular interval time series computed by the exact and the appropriate methods, respectively. Based on the triangle inequality, we derive the error bound for the approximate algorithm.

Proposition 11. *Compared to the exact cost $\Delta(\mathbf{t}, \mathbf{s}^{exact})$, Algorithm 4 returns an approximate solution with cost $\Delta(\mathbf{t}, \mathbf{s}^{appr})$, having*

$$\frac{\Delta(\mathbf{t}, \mathbf{s}^{appr})}{\Delta(\mathbf{t}, \mathbf{s}^{exact})} \leq 1 + \frac{8\lambda_d}{\delta_\epsilon^{min}}$$

where $\delta_\epsilon^{min} = \min(|\epsilon_{md} - \epsilon_i|, \lambda_d)$, $i = 1, 2, \ldots, n-1$ and $\epsilon_i \neq \epsilon_{md}$.

As shown, the approximation is bounded by factors of λ_d the delete cost, ϵ_i the intervals of $\mathbf{t}$, and ϵ_{md} the median interval of $\mathbf{t}$.

In the special case that ϵ_T^{exact} and s_{md}^{exact} are exactly ϵ_{md} and t_{md}, we can further prove that the approximate algorithm indeed returns the optimal solution. That is, the median interval and the median timestamp of $\mathbf{t}$ are accurate in this case, exactly those of the optimal repair. It is highly probable given the robustness of the median in handling noisy time series data [29].

Corollary 12. *If $\epsilon_T^{exact} = \epsilon_{md}$, $s_{md}^{exact} = t_{md}$, the approximate algorithm returns the minimum repair cost, $\Delta(\mathbf{t}, \mathbf{s}^{appr}) = \Delta(\mathbf{t}, \mathbf{s}^{exact})$.*

5 SYSTEM IMPLEMENTATION

In this section, we implement the timestamp repair approach as a function `timestamprepair` in an open-source time series database, Apache IoTDB [1, 2]. In addition, the timestamp repair results could be utilized to evaluate the data quality of the time series, including `timeliness`, `completeness` and `consistency`.

5.1 Timestamp Repair Function

We implement the efficient approximate version of timestamp repair, Algorithm 4, as the function `timestamprepair` [4]. It can be used in a SQL statement as follows

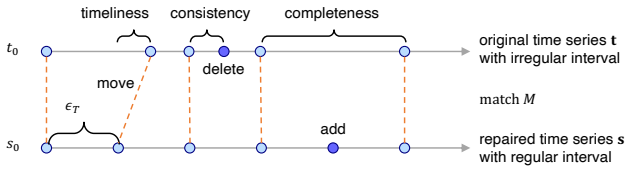

Figure 4: Data quality measures

Table 2: Features of the datasets. Quality issues are the data quality issues existing in the original datasets. Truth denotes the ground truth factors of the dataset that we have.

Dataset	Data quality issues	Truth	#Points
Engine	delayed	ϵ_T, s_0, m	43,954
Turbine	delayed, missing, redundant	ϵ_T	28,000
Vehicle	delayed, missing, redundant	-	111,790
Energy	-	ϵ_T, s_0, m	19,735
PM	-	ϵ_T, s_0, m	43,824
Air Quality	-	ϵ_T, s_0, m	9,357

```
select timestamprepair(s1, [interval]) from root.test.d1
```

where interval is an optional parameter to deal with different scenarios when the interval of the time series is given or not. It returns a repaired time series of `s1` with regular time intervals.

5.2 Data Quality Measures

As introduced in Section 1, the irregular time intervals are introduced by delayed, missing or repeated data points, typical data quality issues. To this end, we can utilize the repair result $\mathbf{s}$ and the match M to profile data quality measures for time series databases. The aforesaid data quality issues lead to three time series data quality measures, including `timeliness`, `completeness`, and `consistency` [5], in the time dimension as illustrated in Figure 4.

Definition 7 (`timeliness`). *Timeliness measures the ratio of the data that are not delayed referring to M, defined by*

$$\texttt{timeliness}(\mathbf{t}) = 1 - \frac{|\{(t_i, s_j) \mid (t_i, s_j) \in M \wedge t_i \neq s_j\}|}{|\mathbf{s}|}. \quad (10)$$

Definition 8 (`completeness`). *Completeness measures the ratio of the data that are not missing referring to M, defined by*

$$\texttt{completeness}(\mathbf{t}) = 1 - \frac{|\{(*, s_j) \mid (*, s_j) \in M\}|}{|\mathbf{s}|}. \quad (11)$$

Definition 9 (`consistency`). *Consistency measures the ratio of the data that are not redundant referring to M, defined by*

$$\texttt{consistency}(\mathbf{t}) = 1 - \frac{|\{(t_i, *) \mid (t_i, *) \in M\}|}{|\mathbf{s}|}. \quad (12)$$

The corresponding SQL statements for these measures are

```
select completeness(s1) from root.test.d1
select consistency(s1) from root.test.d1
select timeliness(s1) from root.test.d1
```

Table 3: $RMSE_f$, $RMSE_t$ and Time cost of RIR-Exact and RIR-Appr compared to the existing approaches

Dataset	$RMSE_f$				$RMSE_t$				Time cost (s)			
	SCREEN	CTTC	RIR-Exact	RIR-Appr	SCREEN	CTTC	RIR-Exact	RIR-Appr	SCREEN	CTTC	RIR-Exact	RIR-Appr
Engine	2.29	0.24	**0.14**	0.73	22.63	6.31	**0.88**	4.34	$\mathbf{1.80 * 10^{-3}}$	$8.91 * 10^{-2}$	0.178	$5.18 * 10^{-2}$
Energy	7.34	5.24	**0.37**	5.00	1833.58	457.15	**2.20**	475.86	$\mathbf{4.28 * 10^{-2}}$	141	154	42.1
PM	6.11	3.58	**0.33**	0.72	2090.94	2.20	**1.80**	4.20	$\mathbf{7.30 * 10^{-2}}$	1470	707	194
Air Quality	6.41	3.98	**0.33**	0.71	2220.79	2.80	**2.00**	2.40	$\mathbf{7.58 * 10^{-2}}$	1770	811	217
Turbine	1.53	1.53	**1.17**	1.53	-	-	-	-	$\mathbf{5.41 * 10^{-2}}$	163	226	55.5

6 EXPERIMENTS

6.1 Experiment Settings

6.1.1 Datasets and Pre-Processing. The datasets and their data quality issues are listed briefly in Table 2. The first three datasets are from our partner companies, where the Engine dataset has the ground truth timestamps, whereas the Turbine data are known to be collected in every 7s but with unknown start and length.

For datasets with regular interval timestamps, including Energy, PM and air quality, we first inject random delays into the timestamps based on Gaussian distribution. The missing points and redundant points are then introduced, controlled by the error rate. Without further statement, the error rate is set to 1%, i.e., 1% repeated data are injected into the data and 1% of the missing points are selected to be removed.

6.1.2 Metrics. Given the repaired result $\mathbf{s}^r$ and the truth $\mathbf{s}^t$, we consider two metrics, including RMSE loss between the time series ($RMSE_t(\cdot)$) and RMSE between the factors ($RMSE_f(\cdot)$).

RMSE loss between the time series ($RMSE_t(\cdot)$) evaluates the difference on (repaired and truth) timestamps. We consider RMSE over the shorter length $\min(|\mathbf{s}^r|, |\mathbf{s}^t|)$ length of both sequences, i.e., $RMSE_t(\mathbf{s}^r, \mathbf{s}^t) = \sqrt{\frac{\sum_{i=0}^{\min(|\mathbf{s}^r|,|\mathbf{s}^t|)}(s_i^r - s_i^t)^2}{\min(|\mathbf{s}^r|,|\mathbf{s}^t|)}}$.

Recall that the interval, start and length uniquely determine all the timestamps of regular interval time seris $\mathbf{s}$. Thereby, RMSE between the factors ($RMSE_f(\cdot)$) evaluates the regular interval time series through the three factors. Let ϵ_T^r, s_0^r, m^r denote the three factors of $\mathbf{s}^r$ and ϵ_T^t, s_0^t, m^t denote the three factors of $\mathbf{s}^t$. We have $RMSE_f(\mathbf{s}^r, \mathbf{s}^t) = average(w_\epsilon|\epsilon_T^r - \epsilon_T^t|, w_s|s_0^r - s_0^t|, w_l|m^r - m^t|)$ where w_ϵ, w_s and w_l are weights for three factors. Since the interval, start and length factors are in different scales, we normalize the differences of three factors and take the average according to their weights, e.g., $(w_\epsilon, w_s, w_l) = (1, 0.5, 0.1)$.

6.1.3 Baselines. We include CTTC [28] and SCREEN [29] as baselines. CTTC cleans timestamps under temporal constraints in a form of temporal networks [13]. SCREEN cleans time series by sliding windows, based on speed constraints, i.e., the variance of the values. We adapt SCREEN to timestamp repairing by regarding timestamps as values. For both CTTC and SCREEN, we use the median of the time intervals in $\mathbf{t}$ as the input constraints.

The experiment code and public datasets are available at [6].

6.2 Comparison with Existing Methods

For each dataset, we compare our proposal with the existing methods on $RMSE_t$, $RMSE_f$ and time cost. As reported in Table 3, RIR-Exact outperforms other methods in terms of $RMSE_t$ and $RMSE_f$ over all the datasets. The result is not surprising, since it searches exactly for the minimum cost of modification, as presented in Section 1. For the same reason, RIR-Appr also shows stably competitive results in both metrics, but lower time cost than RIR-Exact.

Since SCREEN is conducted based on sliding windows, which fails to learn an overall repair, thus showing worse RMSE results than other approaches, and of course lower time cost.

CTTC employs temporal constraints to repair the timestamps, which fails to consider the insertion and deletion of the points. In most datasets, it performs worse than RIR-Appr except Engine dataset, which contains only delayed points.

6.3 Application Case Studies

6.3.1 Frequency-Domain Analysis. To apply timestamp repairing to FFT, while the values of delayed and deleted points are naturally processed, for the missing points, we need to impute the corresponding values, e.g., by linear interpolation [17], as introduced in Section 1. Once the missing values are also imputed, we can compare the time series repaired (on both timestamps and values), in terms of FFT results in two aspects: (1) the time cost of applying frequency-domain analysis over different sequences, and (2) RMSE loss between the analyzed results of various methods and the results of ground truth. That is, we evaluate the distance of the repaired time series to the ground truth in the frequency domain. Lower RMSE means better performance. NUFFT [14] with built-in interpolation is also reported as a baseline.

Figure 5(a) reports the time cost of the repaired time series with frequency-domain analysis approaches over Energy dataset. For original data with irregular intervals, only NUFFT could be applied, thereby showing the largest time cost. Time costs of other sequences with regular intervals are close, demonstrating the advantage of regular interval sequences. Moreover, Figure 5(b) further shows the RMSE results. It is not surprising that our RIR-Exact shows the best performance, analogous to the results illustrated in Table 3. Besides, RIR-Appr obviously outperforms other baseline methods in RMSE, again verifying the superiority of our approximate repair.

6.3.2 Data Compression. Another benefit of regular interval time series is the nature of being compressed, especially in the time series database that employs second-order differences for compression. We therefore implement the algorithm in an open-source time series database Apache IoTDB [30], and leverage the database to store the original and repaired time series respectively.

In this experiment, we use the timeseries database to store original and repaired time series data from 0.6 to 3 million data points. For each size, we insert data into the database and record the increases of the storage space. Figure 6(a) reports the results on

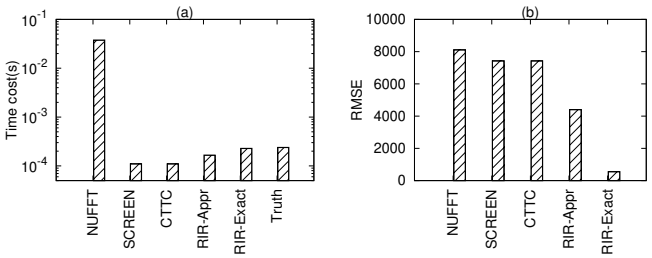

Figure 5: Application on frequency-domain analysis

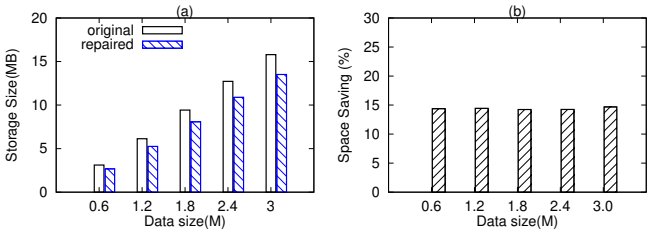

Figure 6: Application on data compression

Table 4: Results of the data quality measures over different real-world use cases.

Dataset	`timeliness`	`completeness`	`consistency`
Engine	0.988	1	1
Turbine	0.899	0.718	0.853
Vehicle	0.955	0.751	0.984

the storage size. It is not surprising that the repaired time series occupy less space in the database. We also report the spacing saving ratios between the methods in Figure 6(b), computed by $Space\ Saving\ Ratio = 1 - \frac{Space_{repaired}}{Space_{original}}$, where $Space_{repaired}$ and $Space_{original}$ denotes the space used by repaired data and original data, respectively. Due to the principle of the second-order differences, regular interval time series (with 0 second-order difference in timestamp) takes minimal space for storage. As shown in Figure 6(b), the *Space Saving Ratio* is stably around 15%. That is to say, the regular interval timestamps are compressed effectively.

6.3.3 Data Quality Measures. In Section 5.2, upon timestamp repairs, we implement three data quality measures, i.e., `timeliness`, `completeness` and `consistency` in an open-source time series database Apache IoTDB [1]. To demonstrate the applications of data quality evaluation, we report the results from our partner companies, where IoTDB is deployed to manage time series data including Engine, Turbine and Vehicle with real data quality issues as summarized in Section 6.1.1 in Table 2.

Table 4 reports the results of the case study over three real-world use cases. (1) For Engine data, as the truth we obtain ahead (Section 6.1.1), the main quality issue is known as delayed points. The measures thereby show a lower result in `timeliness`, which conforms to the ground truth. (2) The Turbine data are delayed, missing and redundant in different degrees, reflected by `timeliness`, `completeness` and `consistency`, respectively. (3) For Vehicle data, the ground truth of the time series factors are unknown owing to the extremely noisy data. According to the measures, missing data is the main quality issue (also shown in Example 1 (h)).

In summary, the three measures provide a simple but intuitive overview of the time series data quality in the time domain. They do not only provide profiling for the data in the time domain, but could also potentially guide the selection of the data cleaning methods. Moreover, the measures are implemented as SQL statements, which could be simply used for the time series stored in the database.

6.4 Handling Missing Data Points

While this proposal repairs timestamps of data points, one may further employ the existing methods such as [17, 23] for interpolating or imputing the values of the inserted points. In this sense, our method is complementary to any interpolation or imputation methods for handling missing data points.

Let us first introduce how the data points are treated on values in addition to timestamps. To be specific, for the move operation, denoted by (t_i, s_j) in a match M, the data point recorded at t_i is moved to s_j, whose value is not changed. For the delete operation, denoted by $(t_i, *)$, the data point recorded at t_i is deleted, together with the corresponding value. For the insert operation, denote by $(*, s_j)$, a new data point with timestamp s_j is inserted. In such a scenario, any interpolation or imputation methods such as [17, 23] could be applied to impute the missing value for s_j. Since the move and delete operations do not modify the values, errors on values are mainly introduced by the imputation of missing points.

As aforesaid, we combine our exact and appropriate timestamp repair algorithms with the existing data imputation methods, and evaluate the value imputation error (RMSE) of the data points. To be specific, for each missing point $(*, s_j)$ returned in the timestamp repair results, we employ the methods including Interpolation [17], KNN [7] or SoftImpute [23], to impute the missing values. The RMSE loss between the imputed values and the corresponding truths is reported. Thereby, the experiment is conducted over three datasets with ground truth, including Energy, PM and Air Quality.

As shown in Table 5, it is not surprising that RIR-Exact with more accurate timestamp repairs shows better imputation/interpolation performance than those of RIR-Appr in most tests. RIR-Exact with Interpolation shows the best value imputation performance in all the datasets, and thus is recommended for accuracy in practice.

Moreover, Table 6 shows the performance of FFT analysis over the data after repairing both timestamps and values, as introduced in Section 6.3. Again, RIR-Exact with Interpolation, having the best value imputation performance in Table 5, shows the best FFT application performance, in all the datasets.

6.5 Evaluation on Parameter Determination

In experiments, we set the costs λ_a and λ_d to ϵ_{max}. Nevertheless, to verify the effectiveness of setting $\lambda = \epsilon_{max}$, we further conduct experiments by varying λ.

Table 5: Imputation RMSE of combining RIR with different imputation methods over three datasets.

Method	Imputation	Energy	PM	Air Quality
RIR-Exact	Interpolation	**0.237**	**2.326**	**21.461**
	KNN	0.258	3.460	35.144
	SoftImpute	3.812	45.511	69.641
RIR-Appr	Interpolation	0.299	4.367	53.303
	KNN	0.322	4.625	38.568
	SoftImpute	3.709	45.935	73.512

Table 6: FFT RMSE of combining RIR with different imputation methods over three datasets.

Method	Imputation	Energy	PM	Air Quality
RIR-Exact	Interpolation	**60.69**	**9335.25**	**261.39**
	KNN	60.72	9369.60	263.91
	SoftImpute	67.35	9379.25	477.24
RIR-Appr	Interpolation	173.76	10337.05	2805.78
	KNN	173.75	10371.46	2808.84
	SoftImpute	174.11	10392.99	2907.53

Figure 7 reports $RMSE_f$ and time cost under various λ. In addition, we also plot the chosen maximum interval ϵ_{max} with dashed lines. Overall, RIR-Exact and RIR-Appr are robust and not very sensitive to λ when λ is large enough ($\lambda > 20$ in all the evaluated datasets). In Energy and Air Quality, $RMSE_f$ of either RIR-Exact or RIR-Appr starts to increase a bit when $\lambda > \epsilon_{max}$. The reason is that as aforementioned, a larger λ could make the deletion and insertion difficult. Another observation from Figure 7 is that the time cost decreases with the growth of λ. This is because a larger λ might introduce less deletions and insertions, thus reducing the length of the match M. According to the evaluation, $\lambda = \epsilon_{max}$ could balance the accuracy and the time cost.

6.6 Evaluation on Various Errors

6.6.1 Errors Detectable by Values. In this section, we compare the proposed RIR-Exact-V, considering data characteristics of values, with RIR-Exact, which only considers timestamps. To fully evaluate the consideration of data characteristics, we need to simulate the cases illustrated in Figure 3, i.e., to inject errors of the middle points, making the repeated and delayed points hard to distinguish by timestamps only. The injection is as follows. (a) When a point t_i is deleted, we randomly move t_{i-1} to $\frac{t_{i-1}+t_i}{2}$ or move t_{i+1} to $\frac{t_{i+1}+t_i}{2}$. (b) When a point t_i is inserted between t_{i-1} and t_{i+1}, we set $t_i = \frac{t_{i-1}+t_{i+1}}{2}$, and randomly move t_{i-1} to $\frac{t_{i-1}+t_{i-2}}{2}$ or move t_{i+1} to $\frac{t_{i+1}+t_{i+2}}{2}$. Since the solutions share the same repair costs on timestamps, not distinguishable by $RMSE_f$, we use Precision, Recall and F1-score to evaluate the correctly found matches. Let M_t and M' denote the ground truth match and the match computed by the algorithm, $Precision = \frac{|M_t \cap M'|}{|M'|}$, $Recall = \frac{|M_t \cap M'|}{|M_t|}$, and $F1\text{-}score = \frac{2*Precision*Recall}{Precision+Recall}$.

Figure 8 presents the results over the Energy dataset. As shown, RIR-Exact-V achieves higher Precision, Recall and F1-score than RIR-Exact, i.e., benefits from considering the data characteristics of values. When the error rate increases, RIR-Exact-V still maintains high precision. The corresponding recall might decrease due to the growth of the error rate. It is not surprising that the time cost of RIR-Exact-V is higher, to maintain the matrix V.

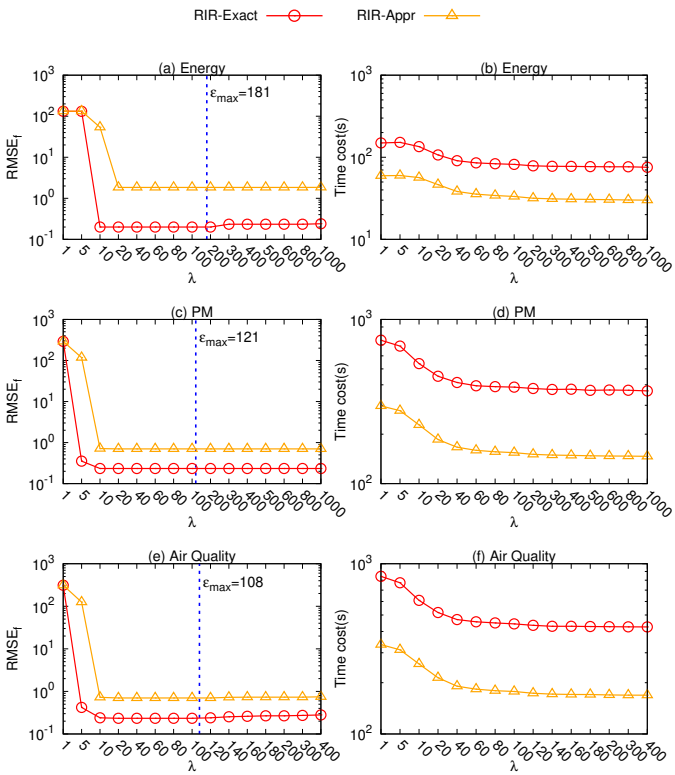

Figure 7: Varying λ over three datasets. The dashed lines are the maximum interval ϵ_{max} of the datasets.

6.6.2 Errors Detectable by Timestamps. To vary the data quality issues of delayed, missing and repeated points, we change the error ratio of each issue, respectively, while fixing the error ratios of the others. The error ratios range from 1% to 10%.

Figure 9 presents the results over the Energy data. When the error ratio increases, the performance of the approximate method decreases, while RIR-Exact still gives accurate results. Compared to the delayed issue, the missing and repeated points affect more the repair, since the number of data points is changed.

7 RELATED WORK

Outliers of timestamps and values are caused by various mechanisms [15]. It is necessary to correct these erroneous timestamps as illustrated in Section 1. However, most existing data repairing techniques (such as [8]) cannot be directly applied due to the distinct difference between temporal constraints and integrity constraints.

CTTC [28] proposes to repair inconsistent timestamps that do not conform to the given temporal constraints. It utilizes a temporal constraint network and proves that any repair can be transformed to a solution with tight chains. Then, CTTC captures a set of candidates via unchanged assignments and tight edges, and determines

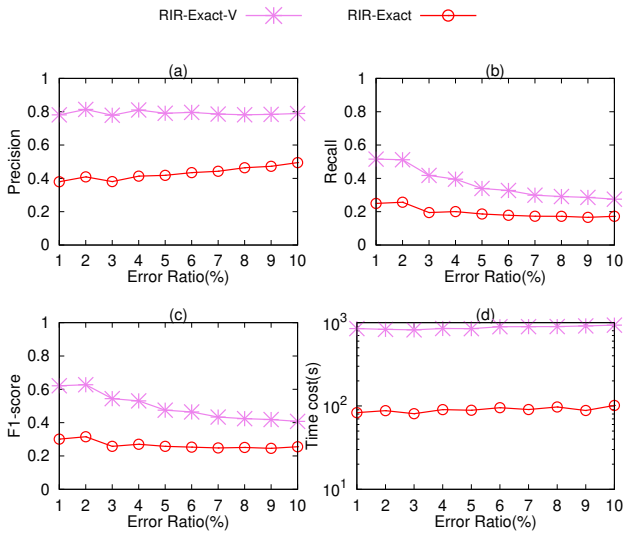

Figure 8: (a) Precision, (b) Recall, (c) F1-score and (d) Time cost of RIR-Exact-V and RIR-Exact over Energy dataset.

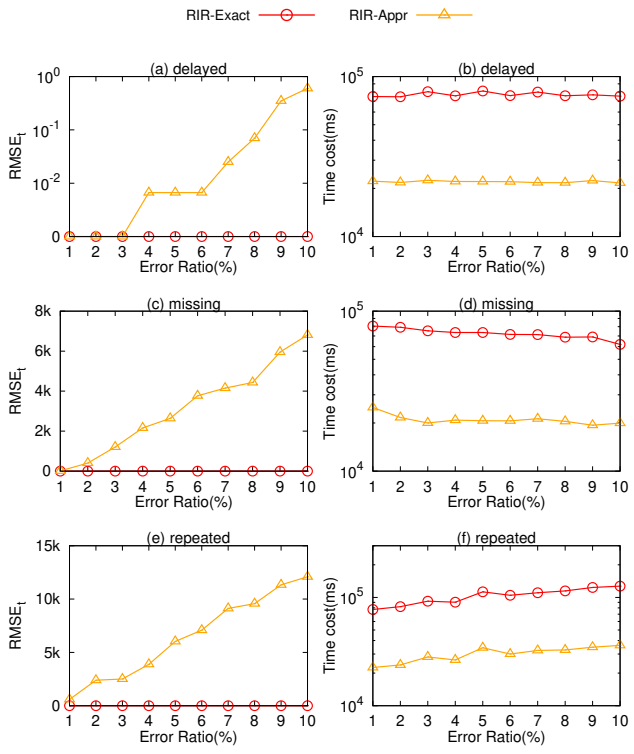

Figure 9: Varying ratios of different data quality issues

the repair with a heuristic algorithm. In our problem setting, the temporal constraint network is a chain, where each node denotes a timestamp and all temporal constraints on edges are the same interval, i.e., $[\epsilon_T, \epsilon_T]$. CTTC is unable to determine the constraints, i.e., the interval ϵ_T should always be given in advance. Moreover, CTTC fails to consider the insertion or deletion of points, which leads to a huge shift when there exist missing or repeated points.

Holistic approach [12] can repair erroneous timestamps by expressing regular time intervals as denial constraints [11]. Similar to CTTC, Holistic repair does not handle missing and repeated points.

Another timestamp repairing algorithm is [26]. Unlike CTTC and Holistic repairing algorithms, [26] is based on probability rather than constraints. It first determines the order of the data points and then adapts the timestamps. However, it is highly dependent on the correctness of the ordering returned by the first step.

SCREEN [29] is a stream data cleaning algorithm. It considers the constraints on the speed of data changes. In our problem setting, the speed constraint is set as the given time interval, to repair the irregular interval timestamps in each window.

Although Algorithm 1 is devised based on dynamic programming, the abstraction of Match Searching is different from a standard edit problem [18] or its variants [10]. The reason is that both the source and target strings are given in the standard edit problem, while the length of our target time series is not fixed and with a dynamic bound for length in Algorithm 1. That is, our approach does not only search for the match M, but also adaptively finds the optimal length for the target time series with the dynamic bound (Lemma 2), whereas the standard edit problem has a known target.

8 CONCLUSION

In this paper, to repair dirty timestamps in a time series for regular time intervals, we propose to move, insert and delete data points. The repairing problem has multiple interacting factors to determine, including the start, length and interval of the time series, if not given in advance, making it challenging. We investigate the lower bounds of repairing for efficient pruning. Moreover, an approximation based on bi-directional dynamic programming is also devised for more efficient repair. Comprehensive experiments demonstrate the superiority of our proposals in both repair accuracy and downstream applications, e.g., frequency-domain analysis and data compression.

To repair timestamps for irregular interval time series, one may consider predicting the next timestamp using the previous ones by a machine learning model [31]. The problem, however, is more complicated. Rather than a regular time interval to determine in this study, it is difficult to decide how distant the prediction deviates from the observation as a timestamp error. Another direction to improve the paper is to further combine the data values with the problem, e.g., try to incorporate data characteristics in the repair costs, rather than only utilizing them to decide matches in Section 3.1.2. As data characteristics, which vary in datasets, are more complicated to consider than timestamps, we thus leave this interesting yet challenging problem as a future study.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (62072265, 62021002), the National Key Research and Development Plan (2021YFB3300500, 2019YFB1705301, 2019YFB1707001), BNR2022RC01011. Shaoxu Song is the corresponding author.

REFERENCES

[1] Apache IoTDB. https://iotdb.apache.org. Accessed May 2022.

[2] https://github.com/apache/iotdb/tree/master/library-udf. Accessed May 2022.

[3] https://sxsong.github.io/doc/timestamp.pdf. Accessed May 2022.

[4] https://iotdb.apache.org/UserGuide/Master/Library-UDF/Data-Repairing.html. Accessed May 2022.

[5] https://iotdb.apache.org/UserGuide/Master/Library-UDF/Data-Quality.html. Accessed May 2022.

[6] Github Repository. https://github.com/fangfcg/regular-interval-repair. Accessed May 2022.

[7] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[8] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD Conference*, pages 143–154. ACM, 2005.

[9] X. Cao, G. Cong, and C. S. Jensen. Mining significant semantic locations from GPS data. *Proc. VLDB Endow.*, 3(1):1009–1020, 2010.

[10] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In F. Özcan, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 491–502. ACM, 2005.

[11] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *Proc. VLDB Endow.*, 6(13):1498–1509, 2013.

[12] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469. IEEE Computer Society, 2013.

[13] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991.

[14] J. A. Fessler and B. P. Sutton. Nonuniform fast fourier transforms using min-max interpolation. *IEEE Trans. Signal Process.*, 51(2):560–574, 2003.

[15] D. M. Hawkins. *Identification of Outliers*. Monographs on Applied Probability and Statistics. Springer, 1980.

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[17] M. Lepot, J.-B. Aubin, and F. H. Clemens. Interpolation in time series: An introductive overview of existing methods, their performance criteria and uncertainty assessment. *Water*, 9(10):796, 2017.

[18] V. I. Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[19] W.-C. Lin and C.-F. Tsai. Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*, 53(2):1487–1509, 2020.

[20] 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.

[21] L. Martí, N. S. Pi, J. M. Molina, and A. C. B. Garcia. Anomaly detection based on sensor data in petroleum industry applications. *Sensors*, 15(2):2774–2797, 2015.

[22] S. E. Marx, J. D. Luck, S. K. Pitla, and R. M. Hoy. Comparing various hardware/software solutions and conversion methods for controller area network (can) bus data collection. *Computers and Electronics in Agriculture*, 128:141–148, 2016.

[23] R. Mazumder, T. Hastie, and R. Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *J. Mach. Learn. Res.*, 11:2287–2322, 2010.

[24] Y. Qin and L. Yang. Throughput comparison of automatic repeat request assisted butterfly networks. In R. C. de Lamare, P. D. Mitchell, M. Haardt, Y. V. Zakharov, and A. G. Burr, editors, *Proceedings of the 2010 7th International Symposium on Wireless Communication Systems, ISWCS 2010, 19-22 September 2010, University of York, York, UK*, pages 581–585. IEEE, 2010.

[25] K. R. Rao, D. N. Kim, and J. J. Hwang. *Fast Fourier transform: algorithms and applications*, volume 32. Springer, 2010.

[26] A. Rogge-Solti, R. Mans, W. M. P. van der Aalst, and M. Weske. Improving documentation by repairing event logs. In *PoEM*, volume 165 of *Lecture Notes in Business Information Processing*, pages 129–144. Springer, 2013.

[27] R. Smolenski, J. Bojarski, A. Kempski, and P. Lezynski. Time-domain-based assessment of data transmission error probability in smart grids with electromagnetic interference. *IEEE Trans. Ind. Electron.*, 61(4):1882–1890, 2014.

[28] S. Song, R. Huang, Y. Cao, and J. Wang. Cleaning timestamps with temporal constraints. *VLDB J.*, 30(3):425–446, 2021.

[29] 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.

[30] C. Wang, X. Huang, J. Qiao, T. Jiang, L. Rui, J. Zhang, R. Kang, J. Feinauer, K. Mcgrail, P. Wang, D. Luo, J. Yuan, J. Wang, and J. Sun. Apache iotdb: Time-series database for internet of things. *Proc. VLDB Endow.*, 13(12):2901–2904, 2020.

[31] S. Xiao, J. Yan, M. Farajtabar, L. Song, X. Yang, and H. Zha. Joint modeling of event sequence and time series with attentional twin recurrent neural networks. *CoRR*, abs/1703.08524, 2017.

[32] X. Yi, J. Zhang, Z. Wang, T. Li, and Y. Zheng. Deep distributed fusion network for air quality prediction. In Y. Guo and F. Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 965–973. ACM, 2018.

[33] H. Yuan and G. Li. A survey of traffic prediction: from spatio-temporal data to intelligent transportation. *Data Sci. Eng.*, 6(1):63–85, 2021.