MTSClean: Efficient Constraint-based Cleaning for Multi-Dimensional Time Series Data

Xiaoou Ding
dingxiaoou@hit.edu.cn
Harbin Institute of Technology

Yichen Song
22S003013@stu.hit.edu.cn
Harbin Institute of Technology

Hongzhi Wang*
wangzh@hit.edu.cn
Harbin Institute of Technology

Chen Wang
wang_chen@tsinghua.edu.cn
Tsinghua University, China

Donghua Yang
yang.dh@hit.edu.cn
Harbin Institute of Technology

ABSTRACT

The widespread existence of time series data in information systems poses significant challenges to data cleaning due to its quality issues, particularly the complex interdependencies among attributes and the persistence of errors. Existing semantic constraints, such as conditional regression rules and speed constraints, though helpful, remain insufficient for this task. This paper introduces two novel online cleaning methods: MTSClean and MTSClean-*soft*, designed to improve cleaning efficiency and robustness. By combining row and column constraints, we significantly accelerate the cleaning process, reducing the time complexity of the exact solution MTSClean from $O\big((NM)^{3.5}|\Sigma|\big)$ to $O\big(NM^{3.5}|\Sigma|\big)$. Meanwhile, MTSClean-*soft* achieves $O\big(NM^2\big)$ and more precise repairs through optimized search for key cells and a novel repair cost function. Comparative experiments against nine benchmark methods highlight our approach's superiority in multiple metrics, completing cleaning tasks faster and performing better than state-of-the-art methods. This demonstrates the practicality and advantage of the proposed methods in cleaning multidimensional time series data.

PVLDB Reference Format:
Xiaoou Ding, Yichen Song, Hongzhi Wang, Chen Wang, and Donghua Yang. MTSClean: Efficient Constraint-based Cleaning for Multi-Dimensional Time Series Data. PVLDB, 17(13): 4840 - 4852, 2024.
doi:10.14778/3704965.3704987

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/dx0o0/DataQualityGroup-MTSClean.

1 INTRODUCTION

With the rapid development of the IoT technology, time series data has become a widely prevalent data type in various fields, referring to a sequence of measurements taken at regular (typically fixed) intervals from measurable quantities in the physical world [11, 32, 40]. Modeling and analytical computations on time-series data have yielded valuable insights. However, it is crucial to acknowledge that real-world time series data often encounters significant data quality problems. In the industrial field, for instance, data loss or distortion due to equipment failures is a common occurrence when sensor data is recorded in databases [40]. Research indicates that industrial time series data frequently harbors over 20% of erroneous records [11, 36]. Even in the finance industry, notorious for its stringent data quality standards, a substantial amount of erroneous time series data exists. For example, stock information on Yahoo Finance has a 93% accuracy rate, while Travelocity's airline data stands at only 95% accuracy [22].

The prevalence of dirty data can significantly hinder downstream time series data analysis tasks, including the training and prediction of machine learning models, by causing them to learn incorrect patterns, thus *degrading performance*. This problem has attracted notable attention both in database [26] and machine learning [21, 30] fields. For instance, inaccurate historical weather data can undermine the reliability of forecasting. Furthermore, decisions based on such data can yield undesirable outcomes, such as inadequate or delayed healthcare plans. Besides, using erroneous data for subsequent tasks can result in a substantial waste of resources, especially in large-scale time series data processing and analysis.

From the above, data quality management is also a crucial aspect of time series data management [40]. In this process, we can leverage data cleaning techniques to effectively identify and correct dirty data, thereby eliminating biases introduced by such data. Traditional cleaning techniques for poor-quality time series data primarily include statistical smoothing methods and constraint-based cleaning techniques (as reviewed in our surveys [11, 41]). The former, an automation-oriented approach prevalent in IoT settings, is noted for its simplicity and low cost. However, smoothing methods typically focus solely on the trend and seasonality of time series data, often modifying a significant portion of the original dataset. This can lead to the loss of valuable information in the original, accurate time series data, subsequently diminishing its utility for downstream applications [26]. For the latter, constraint-based methods require user input of domain-specific knowledge, which can be translated into data quality constraints. We can then examine whether the data satisfies these quality constraints to identify dirty data as violations. Subsequently, algorithms are designed to determine which cells should be modified and their respective repaired values, leading to an improved data quality.

However, compared to traditional relational data, time series data possesses unique characteristics, rendering the problem of data quality more complex. On one hand, the interdependencies between attributes in multi-dimensional time series data are more intricate. For example, when collecting working data of induced

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 13 ISSN 2150-8097.
doi:10.14778/3704965.3704987

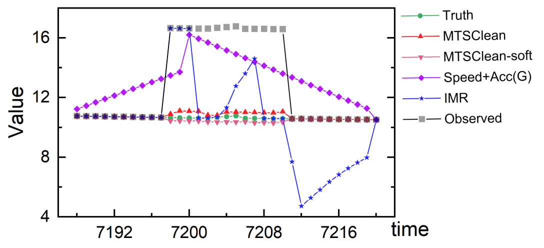


Figure 1: Motivated example demonstration

draft fans, attributes such as mechanical temperature, inlet and outlet air pressure, and power exhibit linear regression relationships. On the other hand, data collected across time points is also interconnected in temporal context.

Recently, semantic constraints capable of describing the quality requirements of time series data have been introduced, including *statistical constraints* [43] that correlate numerical attributes, *speed constraints* [45] describing the rate of change in attribute values over time, and *conditional regression rules* (CRR) [24] expressing regression relationships between attributes. We recently proposed a new notation, TSDD, to capture the contextual relationships inherent in multivariate time series, enriching its semantic representation. Despite advances in expressing quality constraints and mining techniques, time series data cleaning methods based on these constraints are still developing. Example 1.1 demonstrates the current limitations in time series data cleaning effectiveness.

Example 1.1. Figure 1 illustrates a noticeable error record in the water temperature data gathered by a sensor on the induced draft fan over a certain period. This kind of error, known as a continuous error, is frequently encountered in IoT time series data [25, 38]. We present the repair results of potential data cleaning methods. [45] employs speed and acceleration constraints to restrict temperature values to gradually increase without surpassing a predefined threshold. Nevertheless, its repair accuracy is significantly hampered by erroneous data emerging shortly after the initial error. On the other hand, one SOTA cleaning method based on labels, `IMR` [46], is influenced by the number of label parameters and exhibits inconsistent cleaning performance when dealing with continuous errors. Regarding expressive data quality rules, such as CRR and TSDD, effective cleaning strategies have not been introduced yet. However, when applying these expressive constraints into the cleaning process, we achieve notably improved cleaning outcomes, as exemplified by our proposed methods, `MTSClean` and `MTSClean`-*soft*.

Investigating cleaning algorithms supporting complex quality constraints for time series data is urgent. However, several key considerations must be taken into account for constraint-based cleaning, including:

• *Efficiency.* Taking into account the large-scale and high-dimensional nature of time series data, the implementation of multiple complex constraints implies high computational costs, both during the violation detection and repair phases. Consequently, it is imperative to devise highly efficient algorithms to guarantee their applicability to extensive time series datasets.

• *Robustness.* Given the characteristics of time series, the powerful expressiveness of quality rules in describing relationships between attributes and temporal dependencies can lead to complexity in the repair process. More reliable cleaning strategies need to be devised to address issues such as the representation of violation patterns and the identification of more suitable repair objective functions.

• *Online cleaning.* In practical scenarios, such as real-time equipment monitoring and early warning systems, there is a distinct demand for *real-time* or *near-real-time* data cleaning. This enables personnel to respond and make decisions more swiftly regarding the actual issues reflected by poor-quality data. However, the detection complex data errors (*e.g.,* continuous errors, contextual errors) in a timely manner and provide repair solutions within a confined cleaning timeframe present a considerable challenge.

Contribution. Motivated by this, we studied the problem of constraint-based multi-dimensional time series data cleaning. We summarize our contributions as follows:

(1) We formalize the constraint-based cleaning of multi-dimensional time series data, emphasizing linear functional between attributes and temporal constraints, as a linear programming task. Our novel online algorithm, `MTSClean`, significantly reduces cleaning complexity from $O\big((NM)^{3.5}|\Sigma|\big)$ to $O\big(NM^{3.5}|\Sigma|\big)$, where N denotes the length of multivariate time series data, M represents the number of attributes, and $|\Sigma|$ signifies the number of given constraints.

(2) To enhance cleaning efficiency and robustness, we transform the aforementioned linear programming problem into a constrained optimization search. We introduce `MTSClean`-*soft*, a three-stage online cleaning approach (detection, localization, repair), reducing cleaning complexity to $O(NM^2)$. During localization, we refine the traditional hypergraph model for constraint violations in time series, proposing the FindKeyCell algorithm to pinpoint erroneous cells swiftly and precisely by evaluating constraint violations. For repair, we devise a new cost function, balancing minimum repair and constraint violation costs, ensuring robust fixes for intricate error patterns.

(3) We conduct comparative tests on four real-world industrial datasets against nine benchmarks. Results show that `MTSClean` reduced cleaning errors by 40% compared to SOTA cleaning methods, while `MTSClean`-*soft* achieved a 15% reduction with minimal time cost. Ablation studies confirmed that our FindKeyCell boosts the F1-score for violation detection by 0.18 above SOTA.

The paper's **organization** is as follows. Section 2 outlines the cleaning problem. The `MTSClean` algorithm is detailed in Section 3, while Section 4 presents `MTSClean`-*soft* with key theoretical argumentation and examples. Experimental results are reported in Section 5, related works are discussed in Section 6, and Section 7 concludes.

2 PROBLEM OVERVIEW

2.1 Preliminaries

$\mathcal{S} = \{S_1, ..., S_M\}$ contains aligned M-dimensional (a.k.a M-attribute) time series. $attr(\mathcal{S})$ denotes the attribute set of $\mathcal{S}$. For each sequence $S = \langle s_1, ..., s_N \rangle$, $s_n = (x_n, t_n)$, $(n \in 1, 2, ..., N)$ denotes a cell in S, where x_n is a real-valued number with a time point t_n. Below we use $S_i[t_n]$ as the value of sequence S_i at time t_n for short. $w = (l, k)$ denotes a time window containing several consecutive time points from t_l to t_k.

It is recognized that data quality constraints could be classified according to the dependence on attributes (columns in a database table) and instances (rows) [6]. Both constraints are prevalent in time

Table 1: Examples of real data quality constraints

① σ_{row}: $0 \leq (C_2H_2 + C_2H_4 + CH_4 + C_2H_6) \leq 150$
② $\sigma_{row}s$: $C_2H_2 - 0.1C_2H_4 \geq 0, C_2H_2 - 0.1C_2H_4 \leq 3, CH_4 - H_2 \leq 0$
③ σ_{col}: $-0.5 \leq H_2[t_{i+1}] - H_2[t_i] \leq 0.5$
④ σ_{col}: $-0.2 \leq CH_4[t_{i+2}] - 2CH_4[t_{i+1}] + CH_4[t_i] \leq 0.3$

series data. To standardize the representation of quality constraints, we formalize row constraints and column constraints below.

Definition 2.1. Each **row data quality constraint** is represented as a quadruple $\sigma_{row} = (f, \mathcal{A}, f_{min}, f_{max})$, where $\mathcal{A} \subset attr(\mathcal{S})$ represents the sequence set (a.k.a the attribute set) in σ_{row}, f represents a multivariate linear function computed on $\mathcal{A}$. δ: $[f_{min}, f_{max}]$ represent the lower and upper bounds allowed by the computation result of f. Such linear combination of attribute values in $\mathcal{A}$ is expressed as: $f(\mathcal{A}) = \alpha_1 A_1 + \cdots + \alpha_k A_k$, where α represents the *constant coefficient* of each attribute value, $A_i \in \mathcal{A}, (i \in [1, |\mathcal{A}|])$. More specifically, the row constraint could be expressed as follows:

$$\sigma_{row}(t) : \forall t \in T(\mathcal{S}) : (f_{min} \leq \alpha_1 S_{A_1}[t] + \cdots + \alpha_k S_{A_k}[t] \leq f_{max}).$$

It indicates that the subset of attribute values corresponding to data $\mathcal{S}$ must satisfy the restriction imposed by the function f.

Definition 2.2. Each **column data quality constraint** is represented as a quintuple $\sigma_{col} = (f, A, f_{min}, f_{max}, w)$, where A is one attribute from $attr(\mathcal{S})$, and f is the prefined function expressing the temporal dependencies of data values on A in the w-length time window. δ: $[f_{min}, f_{max}]$ represent the bounds allowed by the computation of f. α represents the *constant coefficient*. Specifically, the column constraint $\sigma_{col}(A, w)$ is formalized as

$$\forall t_q - t_1 \leq w : (f_{min} \leq \alpha_1 S_A[t_1] + \cdots + \alpha_q S_A[t_q] \leq f_{max}).$$

Accordingly, a set Σ of expressive data quality constraints for data $\mathcal{S}$ contains both row-constraints Σ_{row} and column-constraints Σ_{col}.

Example 2.3. Using the transformer monitoring scenario in power grid systems as an example, data quality issues can be identified by assessing the conformity of oil chromatographic data with predefined patterns. Table 1 exhibits the semantic expression of practical business needs through defined quality constraints. Specifically, row constraint ① outlines the requirement for total hydrocarbon concentration. The set of row constraints ② elaborate on the "three-ratio method", which involves analyzing gas concentration ratios in transformer oil and comparing them to preset thresholds for diagnosing conditions like electrical discharges. A speed constraint is represented by column constraint ③, indicating that the hydrogen concentration should not fluctuate by more than 0.5 over a certain timeframe. Lastly, ④ introduces an acceleration constraint for CH_4.

2.2 An overview of constraint discovery

Data quality constraints discovery. Our methodology for discovering data quality constraints involves a comprehensive analysis of time series data, referring to SOTA discovery methods from clean data. For row constraints discovery, we traverse each attribute of the data as y in the mapping function $f : X \rightarrow y$ [8, 24]. We then combine the remaining attributes to form the set X and construct all possible mappings f. Using a linear model, we determine the coefficients and biases for each mapping and assess the mapping errors across the dataset. We prioritize mappings based on the minimal error to avoid *redundancy* and ensure comprehensive attribute *coverage* without overlap, repeating the mapping for different attributes sequentially until all attributes are accounted for. For column constraints, we mainly apply speed constraints and acceleration constraint discovery methods in [35] with computing statistical distributions of data in each attribute.

Consistency check. We address potential issues where the constraint set might not provide viable solutions, especially under restrictive conditions combining column constraints with row constraints. By employing a fast linear programming technique at the zero point, we verify the validity of the constraint solution space. This verification is crucial to confirm that all constraints integrated into the repair process are feasible and effective, thereby guaranteeing that the final set of constraints is capable of providing a valid repair solution under all tested conditions.

Implication check. We perform the implication test of constraints using solution spaces. By utilizing the implication axioms demonstrated in our previous works [7, 8, 27], we determine whether the solution spaces of two constraints are in a containment relationship. If the solution space of σ_1 contains that of σ_2, it indicates that the constraint with the smaller solution space has stronger restrictions, thus we delete the constraint with the larger solution space. If the solution spaces of two constraints are not in a containment relationship and there are no conflicts, we merge the two constraints and use the intersection of their solution spaces as the candidate solution space.

2.3 Problem Statement

The repaired version $\mathcal{S}'$ of multi-dimensional time series data $\mathcal{S}$ involves modifying the numerical values of its data points $S_i[t_j]$. To minimize the modification of the original data information, we involve the minimum repair principle which is widely used in data cleaning field to guide the repair process [22, 29]. The repair cost is defined as the norm of the difference sequence between the original data $\mathcal{S}$ and the cleaned data $\mathcal{S}'$:

$$\Delta(\mathcal{S}, \mathcal{S}') = \sum_{i=1}^{M} \sum_{j=1}^{N} \left| S_i[t_j] - S'_i[t_j] \right|$$

Thus, the studied constraint-based cleaning problem are formalized in Definition 2.4.

Definition 2.4. Given time series data $\mathcal{S}$ of length N with M dimensions, and the set $\Sigma = \Sigma_{\text{row}} \cup \Sigma_{\text{col}}$ of data quality constraints that it is required to satisfy, the **global cleaning problem** of $\mathcal{S}$ is to find a repair $\mathcal{S}'$ that $\mathcal{S}'$ satisfies the set Σ and $\Delta(\mathcal{S}, \mathcal{S}')$ is *minimized*, that is,

$$\min \Delta(\mathcal{S}, \mathcal{S}')$$
$$s.t. \; \forall \sigma \in \Sigma : f_{min} \leq f(\mathcal{A}_\sigma) \leq f_{max}$$

LP solution for global cleaning. From the above, the cleaning task focuses on minimizing the distance between the repaired data and the original data while satisfying given constraints. Note that the cleaning cost $\Delta(\mathcal{S}, \mathcal{S}')$ can be transformed into a form that is suitable as the objective function for linear programming (LP) problems. Specifically, by introducing new variables u and v to represent repair costs, let $|S_i[t_j] - S'_i[t_j]| = u_{ij} + v_{ij}$ and $S'_i[t_j] - S_i[t_j] = u_{ij} - v_{ij}$, then it has $\Delta(\mathcal{S}, \mathcal{S}') = \sum_{i=1}^{N} \sum_{j=1}^{M} u_{ij} + v_{ij}$. Accordingly, by equivalently converting quality constraint predicates in Σ into constraints of LP, LP solver can be used to calculate the cleaned data in the global optimal solution. Therefore, the global optimal cleaning problem can be solved in polynomial time $O((NM)^{3.5}|\Sigma|)$ using the interior-point method [45], where NM represents the number of applied cells and $|\Sigma|$ is the number of constraints.

3 LOCAL OPTIMAL CLEANING: MTSCLEAN

We highlight that the aforementioned basic global optimal repair methods are not ideal. Firstly, even with polynomial time complexity, cleaning methods can be time-intensive for practical use. In industrial settings like power grids and manufacturing, time series data often spans over 100K timestamps. Using global optimization for such data means treating each point in the multidimensional time series as a variable, leading the LP solver to handle a large number of variables, thus impacting repair efficiency. Secondly, global optimal repair treats the entire dataset $\mathcal{S}$ as a whole, necessitating the collection of all data before cleaning can commence. To enhance the efficiency of cleaning massive streaming data, we transition from a global to a local optimal repair. This method concentrates solely on recently collected data within a designated time window, preserving already cleaned historical data. Taking into account the distinct characteristics of row and column constraints on data requirements, we formally define the local optimal cleaning problem in Definition 3.1.

Definition 3.1. Given low-quality time series data $\mathcal{S}$ and the set Σ of data quality constraints, the **local cleaning problem** of $\mathcal{S}$ is to find $\mathcal{S}'$ that satisfies the following conditions, with computational complexity not exceeding the upper bound of global cleaning, $O((NM)^{3.5}|\Sigma|)$.

$$\begin{aligned} \min \quad & \Delta(\mathcal{S}, \mathcal{S}') \\ s.t. \quad & \forall \sigma_{row} \in \Sigma_{\text{row}}: && f_{min} \le f(\mathcal{A}, t) \le f_{max} \\ & \forall 0 < t' - t \le w, \sigma_{\text{col}} \in \Sigma_{col}: && f_{min} \le f(A, t, t') \le f_{max}. \end{aligned}$$

As the local repair method retains all quality constraints, it ensures full compliance with cleaning standards. Consequently, compared to global optimal repair, the local approach significantly cuts cleaning time without compromising effectiveness.

3.1 Algorithm MTSClean

We adopt a row-by-row cleaning approach, MTSClean, instead of cleaning the entire dataset. It involves assessing whether each row of data meets a set of constraints and performing data repair accordingly. This effectively reduces the number of variables required in LP problems and enhances the efficiency of the algorithm. Algorithm 1 outlines the local optimal repair process.

MTSClean reads data from the time series database in an ordered manner, *i.e.,* for any $i < j$, the timestamp t_i is less than t_j. According to the repaired data within the sliding window, it determines the range of values for each attribute, $X^{\min}$ and $X^{\max}$, according to column constraints (*e.g.,* ③ in Table 1). Utilizing the column constraints Σ_{col}, the Build function is employed to construct a LP problem. A LP solver is then invoked to compute the optimal solution, establishing candidate repair results for each cell within

Algorithm 1: MTSClean($\mathcal{S}, \Sigma$)

```
Input  : Data $\mathcal{S}$, the set of constraints $\Sigma$.
Output : The repaired version $\mathcal{S}'$ of $\mathcal{S}$.
$\mathcal{S}' \leftarrow$ Copy($\mathcal{S}$);
for $k \leftarrow 1$ to $n$ do
    Compute $X_k^{\min}$ and $X_k^{\max}$ according to $\Sigma_{col}$;
    $LPProblem \leftarrow$ Build($\mathcal{S}, \Sigma, k, X_k^{\min}, X_k^{\max}$);
    $\mathcal{S}'[k] \leftarrow$ LPsolver($LPProblem$);
    $k \leftarrow k + 1$;
return $\mathcal{S}'$
```

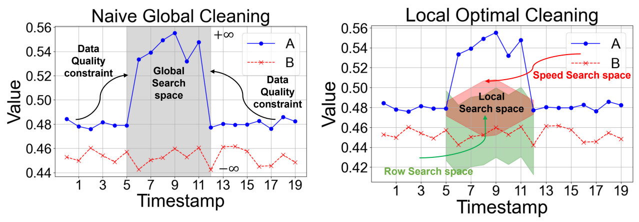


Figure 2: Comparison of search spaces in global and local cleaning

the constraints. During the process, the constraint σ could be transformed into a combination of inequalities as follows:

$$\sigma : (\sum_{i=1}^{k} \alpha_i \mathcal{A}_i \le f^{\sigma}_{max}, \quad \sum_{i=1}^{k} \alpha_i \mathcal{A}_i \ge f^{\sigma}_{min}) \tag{1}$$

This enables the transformation of constraints into combinations of inequality forms, defining a candidate space for the latest data point. This space represents valid repair values that meet the constraints, with any point within it considered clean. Thus, we only need to optimize the modification cost function within the candidate space to obtain the repair result.

Example 3.2. Given a row constraint σ between two sequences A and B in $\mathcal{S}$: $-0.05 \le A - B \le 0.05$, while A experiences a continuous error from t_5 to t_{12}. Figure 2 illustrates the difference in solution space size between MTSClean and the naive global cleaning approach when cleaning this instance.

For data values at t_5, since in LP problem, the values of A and B are treated as variables at each time point, the search space for the repaired value of A theoretically spans from $(-\infty, \infty)$. However, when adopting a locally optimal repair strategy according to Algorithm 1, we first determine the candidate range for the repaired value of A based on column constraints (represented by the red area in Figure 2(b)). Then, we derive the repair range based on the row constraint σ (the green area in Figure 2(b)). The intersection of these two spaces is taken as the candidate repair value range for $A[t_5]$. Evidently, this intersection is significantly smaller than the value space considered by the global cleaning method, highlighting the accelerated search efficiency of MTSClean.

We emphasize to introduce how to improve upon the scenario where local cleaning repairs data to the exact boundary satisfying constraints, by utilizing a novel repair cost function in Section 4.4. This will enable the repaired data to be closer to ground truth within the range that satisfies the constraints.

Proposition 3.3. *The global optimal repair costs $O((MN)^{3.5}|\Sigma|)$, while the local optimal repair* MTSClean *costs $O(N \cdot M^{3.5}|\Sigma|)$ in time.*

Proof. It is acknowledged that the time complexity of LP problem is $O(n^{3.5} \cdot L)$, where n is the number of applied variables and L is the number of constraints in the optimization problem. While for the local optimal repair MTSClean, it limits the LP problem to each row, then the number of variables in the window is M. Compared with global LP, we reduce the size of n from NM to M, so the number of bits encoded by the problem is also reduced by the same proportion. The time complexity for calculating $X_k^{\min}$ and $X_k^{\max}$ using column constraints is $O(1)$, where the calculation is derived from the method Speed+Acc[35]. Thus, MTSClean costs $O(N \cdot M^{3.5}|\Sigma|)$ in time. □

3.2 Theoretical guarantee for MTSClean

As we consider both row and column constraints in the cleaning process, this section discusses their theoretical impact on the repair search space.

3.2.1 Candidate repair space. We extend the candidate range determined by column constraints to multiple dimensions. For each attribute A of $\mathcal{S}$ at timestamp k, it corresponds a candidate range $[x_{k,A}^{\min}, x_{k,A}^{\max}]$. These ranges form a hypercube in high-dimensional space, which we refer to as the *candidate repair space* determined by column constraints *e.g.,* speed constraints and acceleration constraints. Similarly, each row constraint also defines two hyperplanes in high-dimensional space based on its upper and lower bounds. The intersection of the space enclosed by these hyperplanes and the hypercube constitutes the final candidate repair space, determined jointly by the set of row constraints and column constraints.

3.2.2 Theoretical guarantee. We apply Figure 3 to present the search space under both row and column constraints. First, we state in Lemma 3.4 that the optimal solution of the LP problem is equivalent to the optimal solution of the local cleaning problem.

We extract two columns (representing the amplitude in the induced draft fan) A and B with row and column constraints from multivariate time series data (in the diagram, A is represented in black, and B in gray), represent. The red area represents the value range of the same attribute at different timestamps calculated based on velocity constraints starting from any moment t. The blue area represents the value range calculated based on acceleration constraints. The green area represents the value range obtained from row constraints, where our row constraint implies that the difference in values of sequences A and B at the same timestamp should be maintained between -0.03 and 0.05. In the subsequent proof process, we use x to represent the value of sequence A. Lastly, the blue braces represent the candidate range at t_{10}.

Lemma 3.4. *Given data $\mathcal{S}$ and the set of constraints Σ, the repair solution obtained by* MTSClean *for each row in $\mathcal{S}$ is equivalent to the optimal solution of the local cleaning problem.*

Proof. To take speed constraints and acceleration constraints in [35] for example, we can relax the restrictions of speed constraints and acceleration constraints from the global cleaning problem to within a time window of length w. When considering column constraints, instead of any $t_j - t_i \le w$, we focus on the current timestamp t_k and only consider t_j satisfying $t_k - w \le t_j < t_k$ as constraints for t_k. This relaxation is summarized as $[x_k^{\min}, x_k^{\max}]$ described in Section 3.2.1. In Figure 3, this is equivalent to the boundary of the purple region. Our MTSClean, by restricting each attribute's value within its candidate range and then transforming row constraints into constraints for constructing a LP problem, ensures that its optimal solution is equivalent to the optimal solution of the local cleaning problem. □

Next, we prove that the candidate range mapped to each attribute from the candidate repair space is superior to the candidate range obtained by only row constraints or only column constraints.

Lemma 3.5. *The candidate range obtained from the set of constraints Σ is more compact than that obtained from either the row constraints Σ_{row} or the column constraints Σ_{col}.*

Proof. Considering the sparsity of errors, assume that for row constraints, only one column attribute contains an error. Hence, we map the candidate repair space to the potential range of the erroneous attribute as follows. We use the green, blue and red area corresponding to the allowed range of the attribute A which is restricted by row, velocity and acceleration constraint.

If t_k is distant from t_{k-1}, both speed and acceleration constraints yield a broader candidate range [35]. However, row constraints mitigate this by restricting the range, thereby aiding in narrowing the repair options. This effect is not only beneficial at the current timestamp t_k but also continues to benefit subsequent timestamps due to the persistence of speed and acceleration constraints. Row constraints, at the least, do not widen the candidate range; at best, they significantly reduce it. Conversely, when t_k is proximal to t_{k-1}, the range is primarily influenced by column constraint limits.

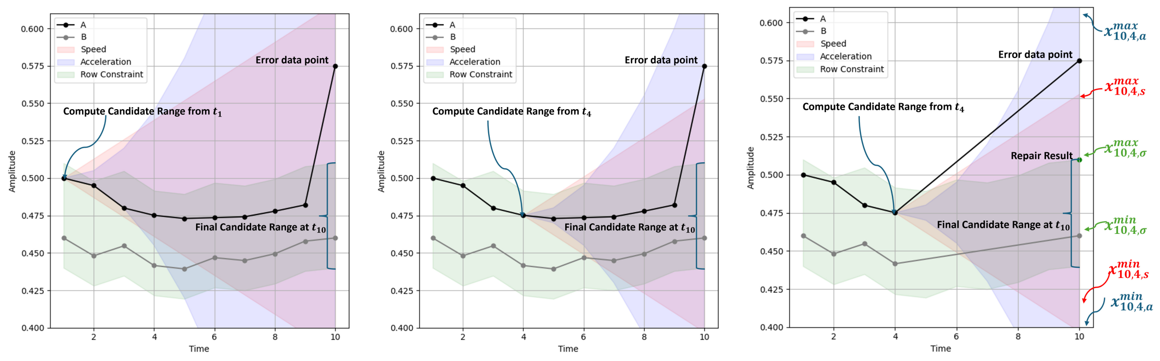
More specifically, Figure 3(b)(c) shows the repair process for attribute A and B. Given a row constraint: $f_{\min} \le \alpha_1 \mathcal{S}_A[t] + \alpha_2 \mathcal{S}_B[t] \le f_{\max}$, it has two cases:
(1). In the case $\alpha_1 \ge \alpha_2$, modifying values on A can meet the row constraint with a smaller modification cost, while A violates the column constraints. Therefore, we do not modify the values on B and can derive the candidate range for A determined by the row constraint (in green). This range may not always fall within $[x_k^{\min}, x_k^{\max}]$. If there is an intersection, the value closest to x_k within the intersection is the repair solution. Otherwise, we can directly use $x_k^{\max}$ as the repair solution. (2). If $\alpha_1 < \alpha_2$, then modifying B can satisfy the row constraint with a smaller modification cost. However, it is first necessary to repair x_k of A to $x_k^{\max}$, then perform the same repair as case (1) for sequence B based on this modification.

From the above, row constraints provide a more compact candidate range on top of column constraints, at least not expanding the cleaning constraint space. Further, when row constraints are not tight enough, column constraints can also provide a more compact candidate range. Hence, combining row and column constraints provides a more optimal range than using either row or column constraints alone. □

Further, we state that the influence of constraints can be propagated forward during the repair process.

Lemma 3.6. *The tightening of the candidate range caused by row constraints has transitivity on the column constraints.*

Proof. Assume that x_k has been cleaned to x_k''. If only column constraints are used, the result would be x_k'. Now we aim to prove that even if x_{k+1} considers only column constraints, the following holds: $[x_{k+1,k''}^{\min}, x_{k+1,k''}^{\max}] \subset [x_{k+1,k'}^{\min}, x_{k+1,k'}^{\max}]$.



(a) The candidate space calculated from t_1 (b) The candidate space calculated from t_4 (c) Row constraint when t not evenly distributed

Figure 3: Demonstration for the repair search space under both row and column constraints

If the row constraint did not play a role in cleaning x_k, then $x''_k = x'_k$ and the equation obviously holds. Therefore, without loss of generality, assume that the row constraint played a role in cleaning x_k, leading to a better repair solution x''_k, such that $x''_k < x'_k (x_k > x^{\max}_k)$. Hence, we can derive $x^{\max}_{k+1,k''}$ and $x^{\max}_{k+1,k'}$ based on the formulas for speed and acceleration constraints:

$$x^{\max}_{k+1,k'',s} = x''_k + s_{\max}(t_{k+1} - t_k), x^{\max}_{k+1,k',s} = x'_k + s_{\max}(t_{k+1} - t_k)$$

From $x''_k < x'_k$, it is evident that $x^{\max}_{k+1,k'',s} < x^{\max}_{k+1,k',s}$. As for the acceleration constraint, we have:

$$x^{\max}_{k+1,k'',a} = (a_{\max}(t_{k+1} - t_k) + \frac{x''_k - x''_{k-1}}{t_k - t_{k-1}})(t_{k+1} - t_k) + x''_k,$$

$$x^{\max}_{k+1,k',a} = (a_{\max}(t_{k+1} - t_k) + \frac{x'_k - x'_{k-1}}{t_k - t_{k-1}})(t_{k+1} - t_k) + x'_k$$

Since $x''_k < x'_k$, both terms in $x^{\max}_{k+1,k',a}$ are smaller than the corresponding terms in $x^{\max}_{k+1,k'',a}$. □

In summary, these lemmas ensure that by simultaneously utilizing row and column constraints during the cleaning process, we can more efficiently and accurately identify the candidate repair space for erroneous data.

4 ALGORITHM MTSCLEAN-SOFT

4.1 Solution Overview

Note that MTSClean constructs LP problems for *all* constraints in Σ and all cells to ensure an accurate search space. However, in practical scenarios, it is unlikely for data to violate numerous constraints simultaneously and rapidly. Our preliminary experiments reveal that, typically, only one to three constraints are violated concurrently during most data quality issues. Given that MTSClean can utilize over dozens of constraints for LP formulation, the optimal solutions for most cells often align with their original values, leading to unnecessary computational efforts. Additionally, the reliance on the minimum modification principle for the optimization objective can yield overly broad repair values for extreme outliers, compromising repair accuracy.

To accelerate the repair process, we focus on reducing both *the number of constraints* really involved in violations and *the number of cells* that actually need to be repaired. Towards this, we introduce an approximate local repair approach, MTSClean-*soft*, detailed in Algorithm 2. This method comprises three phases.

Algorithm 2: MTSClean-*soft*($\mathcal{S}$, Σ)

```
Input  :Data $\mathcal{S}$, the set of constraints Σ.
Output :The repaired version $\mathcal{S}'$ of $\mathcal{S}$.
$\mathcal{S}' \leftarrow$ Copy($\mathcal{S}$);
foreach row in $\mathcal{S}'$ do
    $Vio \leftarrow$ ViolationDetect(Σ);
    $\mathcal{G} \leftarrow$ BuildHypergraph($Vio$);
    $Vio(E) \leftarrow$ getViolationDegree($\mathcal{G}(E)$);
    $\Sigma_{\text{sorted}} \leftarrow$ SortByVio(Σ);
    $OptProblem \leftarrow$ FindKeyCell($\mathcal{S}', \Sigma_{\text{sorted}}, \mathcal{G}$);
    $\mathcal{S}'$[row] $\leftarrow$ UnconstrainedSolver($OptProblem$);
return $\mathcal{S}'$;
```

Step 1: Violation Detection and Constraint Hypergraph Construction. For each data row, we use the ViolationDetect function to assess data integrity, capturing constraint violations and affected cells within a constraint hypergraph $\mathcal{G}$. We employ the BuildHypergraph function to shape the fundamental structure of $\mathcal{G}$ based on violation cells in $\mathcal{S}$. Unlike previous methods like Holistic [5], our approach integrates time-dependent constraints, focusing on constraint violations across time series columns. This enables a single violation hypergraph for all subsequent computations. (see Section 4.2)

Step 2: Key Cell Determination. We introduce a metric to assess the violation degree for each hyperedge in the violation hypergraph, denoted as $Vio(E)$. We then prioritize constraints in Σ based on their violation degree using SortByVio. This helps us pinpoint *key cells* where actual errors likely occurred. To solve this, we apply a minimum vertex cover (MVC) approximation algorithm and devise a heuristic method. This heuristic weighs constraint violation degrees and the number of vertices linked to hyperedges, enhancing key cell identification accuracy. (see Section 4.3)

Step 3: Data repairing. Following the above steps, we repair the identified key cells. Note that we construct a LP problem to solve for the repair values of the vertices output by FindKeyCell function and the sorted constraint in Σ_{sorted}. Specifically, we employ UnconstrainedSolver to solve the unconstrained optimization problem with a well-design objective function (see Section 4.4). This effectively reduces the size of the repairing problem. The process is repeated for each row in $\mathcal{S}'$ until all rows have been processed.

4.2 Violation Detection

Our initial step involves evaluating the data's compliance with constraints Σ. Taking into account the nature of time series data, we mandate two tasks to be fulfilled during violation detection, i.e., (i) *Identification and representation of violated data cells*, and (ii) *Quantification of the degree of constraint violation*. Below we introduce these two parts sequentially.

4.2.1 Constraint hypergraph representation. We introduce *constraint hypergraph* [5] as an effective representation method for characterizing patterns of constraint violations in time series data. Existing constraint hypergraphs do not support the representation of violations involving value constraints on the same attribute across different time points in time series. Therefore, we propose the concept of an intra-window constraint hypergraph in Definition 4.1 to enable the simultaneous representation of both row and column constraints within a time window.

Definition 4.1. The **intra-window constraint hypergraph**, denoted as an undirected graph $\mathcal{G}_w = (V, E)$, consists a set of vertices V and hyperedges E for data in a w-length time window. Each $v \in V$ corresponds to a cell $S_A[t]$ in $\mathcal{S}$. Violated constraints are denoted by hyperedges, formatted as $e \in E = (\sigma; v_1, ..., v_k)$, where k represents the number of cells involved in constraint σ.

We examine the constructed $\mathcal{G}_w$ from both row and column dimensions. $\mathcal{G}_w$ can be treated as a collection of slices, denoted as $P = \{p_1, ..., p_w\}$, where each slice p represents the values of $\mathcal{S}$ at a specific timestamp t within the window w. Hence, row constraints exist within each individual slice, while column constraints link the same attribute across all slices. By arranging all the slices in a planar manner, we obtain the visualized solution representation of the temporal window in Figure 4. In Figure 4(a), each vertex represents a different attribute at the same timestamp, and each hyperedge that covers multiple vertices represents a row constraint, while in Figure 4(b), each hyperedge represents a column constraint, and the two vertices it covers represent the values of the same attribute at two timestamps within a window length w.

Therefore, values of the same attribute at different timestamps correspond to distinct vertices within the intra-window $\mathcal{G}_w$. We put it formally in Proposition 4.2.

Proposition 4.2. *Given $\mathcal{S}$ and Σ, the representation of data violations using the slice set formalism is equivalent to that with an constraint hypergraph representation $\mathcal{G}_w$.*

In this manner, we are able to identify errors by representing them with $\mathcal{G}_w$. The vertex set is directly generated based on the attribute set $attr(\mathcal{S})$ and the window length w. Subsequently, we connect these vertices with hyperedges. For a violated row constraint σ_{row}, we connect the vertices $\{S_A[t] | A \in \mathcal{A}(\alpha)\}$ with the hyperedge $e(\sigma_{\text{row}}; t)$ for every timestamp t in window $[t_1 : t_w]$. For a violated column constraint σ_{col}, we form a hyperedge between the vertex set $\{S_A[t] | t \in [t_1 : t_w]\}$, denoted as $e(\sigma_{col}; t_1, t_w)$.

4.2.2 Violation Degree calculation. After representing violations in $\mathcal{G}_w$, the next important step involves computing the violation degree for each instance. Considering the complex functional mechanism constraints on sequences, it is imperative to quantify the divergence of each violation from the prescribed constraints. This aids us better understand the gravity of data deviations from business requirements, enabling more wide decisions in selecting critical cells for repair. Definition 4.3 formalizes the violation degree.

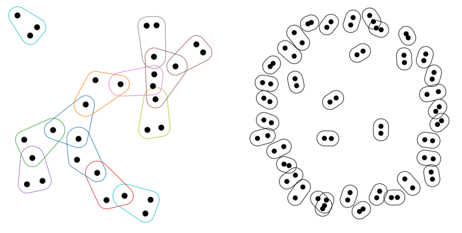

(a) from the row perspective (b) from the column perspective

Figure 4: The slice representation of the intra-window constraint hypergraph

Definition 4.3. For a given constraint σ and the set of cells involved in the content of σ, *i.e.,* $C(\sigma)$, the **degree of violation** of C *w.r.t* σ is represented as a piecewise function $VioDegree(C, \sigma)$

$$= \begin{cases} \min\left\{|f(C,\sigma) - f_{min}|, |f(C,\sigma) - f_{max}|\right\}, C \not\models \sigma. \\ 0, \qquad C \models \sigma. \end{cases}$$

where $f(C, \sigma)$ represents the value obtained through the function expressed by σ for cell C. $C \models \sigma$ denotes that data in C satisfy σ, while $\not\models$ denotes the violation.

The value of the piecewise function $VioDegree(C, \sigma)$ depending on whether the data in C satisfies σ or not. If a violation of σ occurs in $\mathcal{S}$, the actual values among the data have exceeded the boundaries specified by σ (either below the lower bound or above the upper bound), we stipulate that the minimum value from the boundary should be taken as the representation of the degree of violation. Moreover, the severity of the violation is reflected in the increasing scores of the violation degree function, with more severe violations resulting in higher values.

4.3 Key cell determination

With hyperedges in $\mathcal{G}_w$ representing violation cells, we note that not all cells in these hyperedges are actual erroneous data. Therefore, accurately identifying the cells with actual errors, termed *key cells*, is essential. This identification process forms the backbone of our cleaning approach. We draw inspiration from Holistic [5], which employs the *Minimum Vertex Cover* (MVC) algorithm [17] for constraint-based relational data cleaning. However, the MVC algorithm employed by Holistic is a k-approximation algorithm (where k denotes the maximum number of vertices connected by a hyperedge in the hypergraph). It relys on random hyperedge selection and vertex cover set generation with a size ratio of $\frac{|VC_k|}{|VC^*|} \leq k$ compared to the optimal solution.

In practice, a k approximation ratio might not be acceptable. We have explored SOTA MVC approximation algorithms for hypergraphs like `Shuffle` and `MaxDegree` [17], offering an improved approximation ratio of $1 + \lg_2 |V|$ at an average time of $O(|V|^2 |E|)$. By leveraging the unique characteristics of time series data quality issues, we further identify key cells through weighted hyperedge assignments, enhancing search efficiency and effectiveness.

4.3.1 Priority Weighting of Hyperedges. We prioritize hyperedges in $\mathcal{G}_w$ based on the severity of constraint violations, quantified as edge weights. Our selection criteria consider both the edges with the most intersections with other edges and the *degree of constraint violation* for each edge. Edges with higher violation degree are more likely to contain cells that require correction. Similarly, edges

Algorithm 3: FindKeyCell($\mathcal{S}, \Sigma, \mathcal{G}_w$)

```
Input   : $\mathcal{S}$, $\Sigma$, $\mathcal{G}_w$.
Output  : the unconstrained optimization problem.
while $\Sigma \neq \emptyset$ do
    $\sigma_{\max} \leftarrow$ PopMax($\Sigma$);
    UpdateObjectiveFunction($\mathcal{S}$, $\sigma_{\max}$, $Cost$);
    UpdateHypergraph($\mathcal{G}_w$, $\sigma_{\max}$);
$OptProblem \leftarrow$ Construct($\mathcal{S}$, $Cost$);
return $OptProblem$;
Function UpdateHypergraph($\mathcal{G}_w$, $\sigma_{max}$):
    foreach $u \in \sigma_{max}$ do
        foreach $\sigma \in F(u)$ do
            $\mathcal{G}_w \leftarrow \mathcal{G}_w \backslash \sigma$;
    $\mathcal{G}_w \leftarrow \mathcal{G}_w \backslash \sigma_{\max}$;
```

with more intersections are also more likely to represent actual violations. Specifically, for an edge e in $\mathcal{G}_w$, its priority is given by

$$weight(e) = VioDegree(e) + \frac{\sum_{v \in e} |d(v)|}{\max_{v \in e} |d(v)|},$$

where $VioDegree$ calculates the constraint violation degree for a hyperedge, and $d(v)$ denotes the degree of a vertex v in e, *i.e.,* the number of edges connected to v.

We normalize the components in $weight(e)$. For $VioDegree$, normalization involves computing the shortest distance from a linear function to constraint bounds and scaling based on the extreme values from the violation function across all constraints. As the minimum vertex degree in a hypergraph is inherently 0, we forego minimum value retention for vertex degrees.

4.3.2 Algorithm FindKeyCell. After calculating the vertex weights, we prioritize the vertices for repair in descending order of their weights. Algorithm 3 is proposed to identify the key cells. The algorithm initially selects the most significant constraint, denoted as $\sigma_{\max}$, from the non-empty set Σ using the `PopMax` function. It then updates the objective function based on $\mathcal{S}$ and $\sigma_{\max}$, adjusting the associated costs. Simultaneously, the `UpdateHypergraph` function modifies $\mathcal{G}_w$ by iterating over each vertex u within $\sigma_{\max}$ and removing related constraints using the F function. This includes the elimination of constraints directly connected to vertex u and $\sigma_{\max}$ itself from the hypergraph. Upon processing all constraints in Σ, the algorithm constructs the final $OptProblem$ based on the updated $\mathcal{S}$ and the total accumulated cost. The formulated unconstrained optimization problem, $OptProblem$, is returned as the output.

FindKeyCell serves two crucial roles: firstly, it identifies the cells that genuinely require repair and effectively reduces the scale of subsequent problem-solving by pruning cells that are not inherently erroneous. Secondly, for each hyperedge, the calculation of their degree of violation provides more reliable guidance for the construction of the repair cost model in the following step.

4.4 Repairing with the designed cost function

We introduce the cost function for repair operations, emphasizing that merely pinpointing the cells needing repair is inadequate. Determining suitable repair values demands further assessment, which is a non-trivial task. This is because, unlike traditional relational data repair, the repair scope for time series is broader and more intricate. If all erroneous data were only corrected to the constraint set's boundary, the overall outcomes would significantly diverge from the ground truth. Hence, during the repair, we must consider the extent of deviation from constraints, rather than just a binary assessment of whether data meets constraints or not (refer to Figure 2). Consequently, we propose a novel objective function for time series data repair in Definition 4.4.

Definition 4.4. The objective function of cleaning $\mathcal{S}$ w.r.t Σ is $Cost(\mathcal{S}, \mathcal{S}') = \min \Delta(\mathcal{S}, \mathcal{S}') + \sum_{s \in \mathcal{S}'} dist(s, \Sigma)$, where $\Delta(\mathcal{S}, \mathcal{S}')$ is the norm repair cost presented in Section 2.3, $dist()$ represents the repair cost constructed based on the plane bounded by the set Σ of the constraints, denoted as

$$dist(s, \Sigma) = \sum_{\sigma \in \Sigma} \big(\lambda_{lb} \cdot \frac{1}{1 + e^{-s \cdot n_{lb}}} + \lambda_{ub} \cdot \frac{1}{1 + e^{-s \cdot n_{ub}}}\big).$$

This calculation involves summing two components for each σ in Σ. The first component represents the distance of data point s to the lower bound of σ, approximated by the product of s and the lower bound normal vector n_{lb}, transformed via a *sigmoid* function, and weighted by λ_{lb}. Similarly, the second component involves the upper bound, using n_{ub} and λ_{ub}. Specifically, $s \cdot n_{lb} = f_{min} - f(s)$, $s \cdot n_{ub} = f(s) - f_{max}$. Due to *sigmoid* function properties, the final repair remains within the search space defined by Σ.

Consequently, compared to only repairing to boundary values that exactly satisfy the constraints, we achieve a more fine-grained repair. This precise repair offers two main advantages: 1) increased robustness of the repair algorithm, which yields more reliable repair results even when the given constraints have insufficient accuracy (*e.g.,* the allowable deviation threshold δ: $[f_{min}, f_{max}]$ for σ is set large), and 2) a more accurate "restoration" of the ground truth, which has the potential to ensure the reliability of downstream data analysis tasks based on such repair results [23, 26, 30].

Proposition 4.5. *`MTSClean`-soft costs $O(NM^2)$ in time.*

Proof. In `MTSClean`-*soft*, the initial copying of original data takes $O(NM)$ time. Each data row is then processed iteratively. Violation detection during this process requires $O(N|\Sigma|)$. Hypergraph construction involves creating a Boolean matrix of vertices and edges, costing $O(NM|\Sigma|)$ time. Sorting the hyperedges in the graph demands $O(N|\Sigma| \log |\Sigma|)$. FindKeyCell, an approximation for MVC, is called with a time complexity of $O(N|\Sigma| \log |\Sigma|)$. Finally, the unconstrained problem is solved within $O(NM^2)$. Given that $N \gg M \approx |\Sigma|$, the overall time complexity of MTSClean-*soft* is dominated by the highest order term, resulting in $O(NM) + O(N|\Sigma|) + O(NM|\Sigma|) + O(N|\Sigma| \log |\Sigma|) + O(NM|\Sigma|) + O(NM^2) = O(NM^2)$. □

4.5 Comparison of `MTSClean` and `MTSClean-soft`

Table 2 summarizes the similarities and differences between `MTSClean` and `MTSClean`-*soft* from cleaning efficiency and cleaning accuracy. `MTSClean`-*soft* has an advantage over `MTSClean` in cleaning efficiency. `MTSClean`-*soft* costs $O(NM^2)$, and before constructing the optimization problem, `MTSClean`-*soft* uses an MVC strategy to estimate the set of attributes that need cleaning and the data quality constraints should be used. This allows `MTSClean`-*soft* to construct a smaller optimization problem compared to MTSClean, which directly uses all attributes and constraints.

In terms of cleaning accuracy, both MTSClean and MTSClean-*soft* have their advantages. MTSClean emphasizes minimizing modification costs and strictly requires the repaired data to adhere to all row and column constraints. Therefore, in scenarios with guaranteed constraint quality, MTSClean can achieve higher precision cleaning results. On the other hand, MTSClean-*soft* strives to find repair solutions within the constraint limits, additionally considering the degree of constraint violation as a repair cost. Therefore, in scenarios where constraint quality is difficult to guarantee, MTSClean-*soft* is more likely to obtain repair solutions close to the true values within the constraints. However, since MTSClean-*soft* estimates the attributes that need cleaning beforehand, it may occasionally result in incorrect repairs in some cases, whereas the strictness of MTSClean ensures that incorrect repairs do not occur.

5 EXPERIMENTS

5.1 Experimental setting

All programs are implemented in Python. Experiments were performed on a server with 3.70GHz CPU and 64 GB RAM. We have developed a data cleaning system, Clean4TSDB [9, 10, 12], which connects to the time series database Apache IoTDB [39, 40] and implements the cleaning algorithm proposed in this paper.

Experimental dataset. We utilize four real datasets: IDF, SWaT, WADI, and PUMP, as outlined in Table 3. IDF captures daily data on induced draft fans from a power plant's historical database between 2017-01 and 2017-05, amounting to over 1000k timestamps. Its features comprise total power, outlet gas pressure, temperature, inlet gas flow, coil temperature, and more. SWaT, gathered from an industrial water treatment facility, involves parameters such as flow rates, tank levels, and valve statuses, distinctively documenting both regular operations and cyber-attack events. WADI originates from an actual water distribution network, spanning 16 days in 2017. Its records encompass water flow, pressure, and chemical dosages during both normal and attack scenarios. PUMP consists of sensor data monitoring a pump, including metrics like temperature, vibration, and power usage.

Implementation. Since we intercept the clean part of the data, following the same line of precisely evaluating the repair effectiveness, we use the additive Gaussian white noise to synthetically generate noisy data provides a realistic simulation of errors encountered in real-world scenarios [35]. That is, we randomly sample several attributes to inject errors, and the types of errors include single-point errors, continuous errors, inter-attribute relationship errors, etc. The error rate *erate* = 0.1 denotes that 10% rows are replaced.

Comparison Methods. We compare our proposed MTSClean and MTSClean-*soft* to the benchmark cleaning approaches for time series data, including (1) SOTA constraint-based time series cleaning method Speed [37], and its extended version with acceleration constraints Speed+Acc [35]. We implement both global (G) and local (L) modes. (2) HoloClean [33], a SOTA cleaning tools for general big data. (3) IMR [46], a SOTA time series cleaning method with labels. (3) smoother and filter-based EWMA[18], Median[41], Kalman[41].

Preparation of data quality constraints. All the applied constraints are pre-defined by SOTA discovery methods from clean data. For speed constraints and acceleration constraints, all constraints are pre-defined by observing the speed and acceleration statistical distributions of data according to [35]. For row constraints discovery, we apply CRR [24], which gets linear model with the minimum loss [19, 21], and TSDDiscover [8], which mines accurate functional structure and allowable error bound δ for constraints in time series. All discovery methods are available in our git repository.

For IMR, we appropriately adjust the iteration parameters and precision parameters to suit data with error tolerance in iterations to be 0.0001 and the maximum number of iterations to 10000. For HoloClean, since several SOTA algorithms, e.g., IMR and Speed, have adapted HoloClean for time series data cleaning. We follow these approaches.

Metrics. For the original M-dimensional $\mathcal{S}$ containing N records, let $\mathcal{S}_{\text{repair}}$ and $\mathcal{S}_{\text{truth}}$ be the repaired data and the ground truth, respectively. We apply four kinds of metrics to evaluate the effectiveness of data cleaning methods besides the time cost.

(1). *L1-error* [37], which evaluates the closeness of repair to the ground truth, denoted as: $\Delta(\mathcal{S}_{\text{repair}}, \mathcal{S}_{\text{truth}}) = \frac{\sum_{m=1}^{M}\sum_{n=1}^{N}|S_i[t_j]-S'_i[t_j]|}{M\cdot N}$. The lower the L1 error between the repair and truth value is, the closer (more accurate) the repair is to the ground truth.

(2). Relative Repair Accuracy (*RRA*) [37]: to normalize *L1-error* measures and take the errors in $\mathcal{S}$ into consideration. *RRA* verifies the cleaning effectiveness by $1 - \frac{\Delta(\mathcal{S}_{\text{repair}},\mathcal{S}_{\text{truth}})}{\Delta(\mathcal{S},\mathcal{S}_{\text{truth}})+\Delta(\mathcal{S},\mathcal{S}_{\text{repair}})}$.

(3). *F1-score* $= \frac{2\cdot P\cdot R}{P+R}$, where Precision $P = \frac{\#\text{correctRepairCells}}{\#\text{RepairCells}}$ measures the ratio of the number of cells which are closer to the corresponding ground truth to the number of cells which are changed after cleaning. Recall $R = \frac{\#\text{correctRepairCells}}{\#\text{ErrCells}}$ measures the ratio of the number of cells where the method makes a "good" repair to the number of cells where true error takes place.

(4). Violation rate after repair (*VRate*): To evaluate the ratio of the number of constraints still violated in the repaired data to the number of violated constraints in the original data, denoted as *VRate* $= \frac{\#\text{VioCellsAfterRepair}}{\#\text{ErrCells}}$. The closer *VRate* is to 0, the more the cleaning method focuses on the inter-attribute correlations.

5.2 Overall performance evaluation

Table 4 outlines the performance of the cleaning methods. In evaluating the *L1-error* and *RRA*, both MTSClean and MTSClean-*soft* exhibit superior performance. Basic statistical cleaning techniques fare poorly, while SOTA methods, such as IMR, Speed, and Speed+Acc, despite achieving respectable scores, still lag behind our proposed methods. Interestingly, Holoclean performs less well than SOTA time series-specific cleaners, echoing findings from previous studies [37, 46]. This underscores the importance of tailoring constraint-based cleaning to time series data.

Despite some performance differences, MTSClean-*soft* offers a time cost advantage over MTSClean and IMR. Both MTSClean variants achieve an F1 score above 93% in error detection, highlighting the effectiveness of our combined row and column constraints in ensuring accuracy and minimizing false positives. In terms of runtime, our methods prove significantly faster than most others, except smoothing techniques and certain local algorithms. Moreover, our methods exhibit notably lower constraint violations after cleaning, compared to all other tested methods.

Overall, MTSClean achieves the best cleaning results across multiple metrics within an acceptable time cost. When compared to

Table 2: Comparison of MTSClean and MTSClean-*soft*

Aspect	MTSClean	MTSClean-*soft*
Time Complexity	$O(NM^{3.5}\|\Sigma\|)$	$O(NM^2)$
Data Quality Constraint	Requires high accuracy input constraints.	Aims for near-correct values within δ.
Optimization Objective	Minimizes costs strictly.	Flexibly balances costs and violations.
Error Sensitivity	Highly sensitive, immediate correction needed.	Tolerates some violations for better overall repair.

Table 3: Summary of datasets

Dataset	#*Attrs*	#*Size*	erate
IDF [28]	44	1000k	-
SWaT [31]	51	946k	5.85%
WADI [2]	123	2000k	10.7%
PUMP [44]	38	500k	-

Table 4: Overall performance comparison

	IDF					***SWaT***					***PUMP***					***WADI***				
	L1error ↓	***RRA*** ↑	***F1*** ↑	**Time**(s)	***VRate*** ↓	***L1error***	***RRA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RRA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RRA***	***F1***	**Time**(s)	***VRate***
MTSClean	**0.1249**	**0.975**	**0.97**	112.1	**0.294**	**0.4048**	**0.927**	**0.99**	183.3	**0.002**	**0.6008**	**0.751**	**0.94**	24.3	**0.001**	**0.3591**	**0.942**	**0.98**	250.2	**0.001**
MTSClean-*soft*	0.3219	0.792	**0.97**	79.9	0.413	1.1732	0.753	**0.99**	47.46	0.376	0.6476	0.644	**0.94**	12.5	0.216	0.8783	0.752	**0.98**	64.7	0.372
Speed(L)	1.7797	0.171	0.31	24.7	1.064	2.5708	0.251	0.28	40.65	1.468	1.2335	0.028	0.15	3.81	0.994	1.8612	0.183	0.32	42.9	1.092
Speed(G)	0.9671	0.735	0.67	164.8	1.799	2.7043	0.180	0.41	274.6	1.315	1.2345	0.025	0.14	25.9	0.996	1.2974	0.711	0.69	301.4	1.401
Speed+Acc(L)	1.9543	0.145	0.47	40.4	1.104	2.9594	0.136	0.31	68.1	1.206	1.2459	0.029	0.26	6.83	0.998	1.9876	0.167	0.44	71.9	1.173
Speed+Acc(G)	0.8963	0.767	0.68	339.9	1.769	2.6690	0.199	0.44	609.1	1.303	1.2306	0.032	0.18	59.1	0.994	1.1162	0.782	0.68	631.2	1.356
EWMA	1.6130	0.308	0.22	0.01	1.628	2.4749	0.301	0.22	0.02	1.388	1.388	0.308	0.22	0.02	0.948	1.8239	0.321	0.24	0.02	1.457
Median	1.9710	0.003	0.22	0.01	0.999	2.9935	0.059	0.21	0.02	1.002	1.2507	0.003	0.22	0.04	0.999	1.6123	0.006	0.21	0.02	0.993
Kalman	1.8160	0.147	0.22	147.1	1.318	2.7699	0.147	0.22	243.9	1.183	1.1502	0.150	0.22	23.5	0.971	1.4310	0.154	0.24	289.3	1.275
IMR	0.9579	0.668	0.56	171.2	1.320	1.2535	0.736	0.61	221.8	0.535	0.6911	0.618	0.57	83.43	0.494	0.8835	0.702	0.61	243.7	0.482
HoloClean	1.6900	0.429	0.83	310.3	0.891	2.1408	0.518	0.78	560	0.921	0.9491	0.693	0.82	172.1	0.727	1.1125	0.601	0.81	621.3	0.754

MTSClean, the distinct advantage of MTSClean-*soft* lies in its ability to achieve almost the same repair F1 score while improving computational time by at least 30%. It is important to note that MTSClean-*soft* may slightly underperform MTSClean in terms of metrics like *L1-error* and *RRA*, due to its relaxation of constraint satisfaction. Nonetheless, it still maintains a significant advantage over other cleaning algorithms (e.g., IMR, HoloClean, Speed).

5.3 Evaluation with varying parameters

Below we report on the impact of two crucial parameters on the cleaning effectiveness, *i.e.,* the data volume and the error rate.

***Exp1*: Repairing performance with varying data amount**. Figure 5 evaluates the performance of various algorithms as the data volume increases in the WADI dataset. Notably, MTSClean and MTSClean-*soft* maintain superior cleaning performance as data volume expands, producing repairs closest to the ground truth. Conversely, methods like EWMA, Median, and Kalman alter a significant proportion of data points, leading to poor performance in metrics like *RRA* and *VRate.* HoloClean, despite good error detection and an F1 score ranking third after MTSClean and MTSClean-*soft*, is limited to denial constraints and traditional FDs. This restricts its ability to handle more expressive constraints, such as speed constraints or the row constraints introduced in this paper. Additionally, HoloClean's strength lies in processing string-type data, contributing to its less than optimal repair effectiveness. Speed+Acc(L) relies on speed and acceleration constraints for local data cleaning. However, its effectiveness decreases when dealing with complex, significantly deviating errors. It tends to repair only the initial segment of erroneous data, limited by constraint boundaries, leaving middle segment errors untreated (as exemplified in Example 1.1).

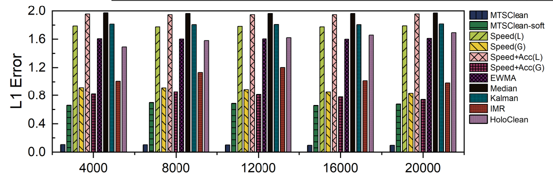
Regarding global methods such as Speed(G), Speed+Acc(G), and IMR, despite showing improved *L1-error* compared to HoloClean, they still fall behind MTSClean and MTSClean-*soft.* While Speed+Acc(G)'s computational approach is inefficient for multivariate time series, IMR relies on external data labeling, using accurate data subsets for repairs. Despite this, its *L1-error* remains inferior to MTSClean. All these methods, including HoloClean, treat multivariate time series holistically, unsuitable for real-time repairs. Conversely, our online algorithms process data row-by-row, demonstrating scalable runtime growth, balancing efficiency and repair effectiveness. In terms of *VRate*, MTSClean and MTSClean-*soft* maintain low constraint violation rates, with stable performance as data volumes increase. This underscores our method's superiority in error detection sensitivity and repair accuracy, with competitive runtime performance.

***Exp2*: Repairing performance with varying error rate**. Figure 6 reports the performance of algorithms on WADI, with a data volume of 20K, when different proportions of rows contain erroneous values. It shows that the repair effectiveness of MTSClean and MTSClean-*soft* does not decline as the error rate increases. MTSClean can maintain high cleaning accuracy when dealing with a large number of errors, further confirming the robustness of the algorithm. L1-error growth of Speed(L), Speed+Acc(L), and smoothing methods is significant, demonstrating their inability to handle cleaning tasks with number of errors. The L1-error growth of Speed(G), Speed+Acc(G), IMR, and HoloClean is relatively slower but still higher than that of MTSClean-*soft.* This is because Speed+Acc(G) do not utilize row constraints compared to MTSClean, leading to insufficient repair effectiveness. IMR can only use limited correct data labels, and as errors increase, the proportion of errors that IMR can repair decreases. HoloClean, on the other hand, consistently exhibits insufficient repair accuracy, resulting in its error increasing with the number of errors.

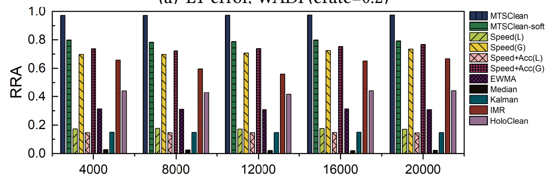
The time cost of MTSClean and MTSClean-*soft* increases with the injection of more errors, which is reasonable. This is because both algorithms need to detect constraint violations for each row of data and only perform cleaning if a violation is detected. Therefore, when there are more errors, the algorithms need to perform more cleaning operations. However, overall, both algorithms maintain a relatively low time cost. Regarding the F1 score, MTSClean and MTSClean-*soft* continue to maintain a clear advantage with stable performance. Note that the F1 scores of EWMA, Median, and Kalman increase as the error ratio increases, but this is a misleading phenomenon. For the three algorithms, their error detection results consistently label almost all rows as erroneous data. Therefore, as more errors increase, F1 scores increase, but this does not indicate that these smoothing algorithms have stronger error detection capabilities.

Table 5: The influence of the accuracy of the input constraints

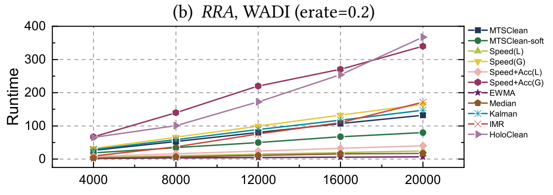
IDF	$\rho = 0.05$					$\rho = 0.5$					$\rho = 5.0$				
	L1error	***RAA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RAA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RAA***	***F1***	**Time**(s)	***VRate***
MTSClean	0.1026	0.973	0.836	23.12	0.298	0.3759	0.895	0.863	25.44	0.187	0.4113	0.883	0.872	26.90	0.159
MTSClean-*soft*	0.9758	0.782	0.836	11.70	0.524	0.9774	0.774	0.863	12.61	0.404	0.9725	0.773	0.872	15.60	0.496
WADI	$\rho = 0.5$					$\sigma = 2.5$					$\rho = 5.0$				
	L1error	***RAA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RAA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RAA***	***F1***	**Time**(s)	***VRate***
MTSClean	0.1443	0.884	0.891	76.24	0.001	0.1831	0.762	0.778	105.64	0.167	0.2208	0.637	0.733	108.97	0.263
MTSClean-*soft*	0.1966	0.814	0.891	14.48	0.288	0.2042	0.661	0.778	26.37	0.331	0.2158	0.648	0.733	25.95	0.416
PUMP	$\rho = 5.0$					$\rho = 6.0$					$\rho = 7.0$				
	L1error	***RAA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RAA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RAA***	***F1***	**Time**(s)	***VRate***
MTSClean	0.6548	0.684	0.858	28.09	0.000	0.8542	0.585	0.839	28.22	0.000	1.0623	0.463	0.812	27.91	0.000
MTSClean-*soft*	0.5271	0.767	0.858	4.36	0.081	0.5674	0.758	0.839	4.31	0.201	0.6095	0.773	0.812	3.90	0.214
SWaT	$\rho = 0.5$					$\rho = 2.5$					$\rho = 5.0$				
	L1error	***RAA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RAA***	***F1***	**Time**(s)	***VRate***	***L1error***	***RAA***	***F1***	**Time**(s)	***VRate***
MTSClean	0.1398	0.892	0.889	60.12	0.002	0.1792	0.768	0.791	102.48	0.178	0.2176	0.645	0.739	105.76	0.264
MTSClean-*soft*	0.1925	0.822	0.889	12.87	0.281	0.1986	0.672	0.791	23.47	0.323	0.2098	0.655	0.739	24.15	0.402



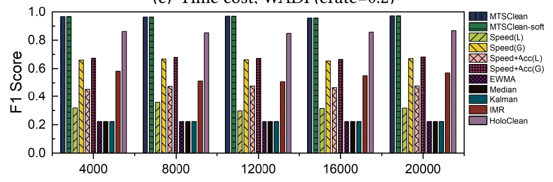
(a) L1-error, WADI (erate=0.2)



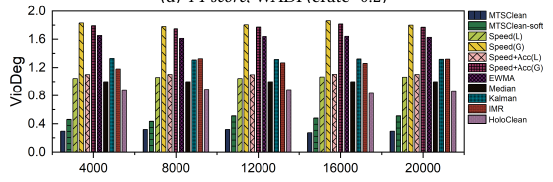
(b) *RRA*, WADI (erate=0.2)



(c) Time cost, WADI (erate=0.2)

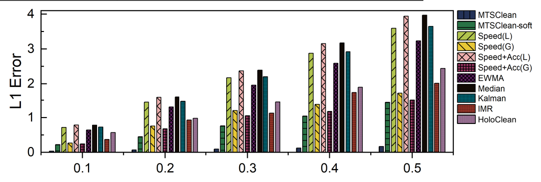


(d) *F1 score*, WADI (erate=0.2)

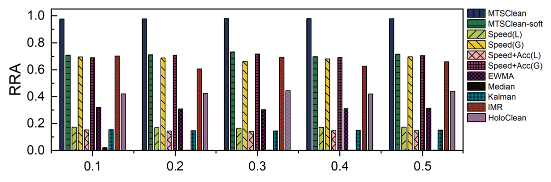


(e) *VRate*, WADI (erate=0.2)

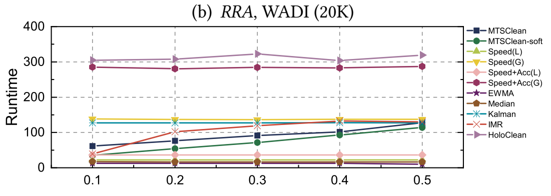
Figure 5: Evaluation with varying data amount



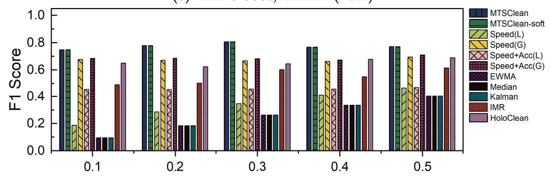
(a) *L1-error*, WADI (20K)



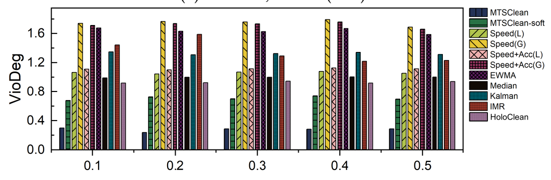
(b) *RRA*, WADI (20K)



(c) Time cost, WADI (20K)



(d) *F1 score*, WADI (20K)



(e) *VRate*, WADI (20K)

Figure 6: Evaluation with varying error rate

This set of experiments fully demonstrates the necessity of simultaneously using multiple types of expressive constraints for multivariate time series data cleaning and the robustness of the proposed cleaning methods.

5.4 Ablation experiment

This section presents two sets of ablation experiments to demonstrate the reliability of the proposed cleaning techniques.

***Exp3*: The influence of the accuracy of the input constraints.** Considering that the accuracy of the input constraint set significantly impacts the cleaning effect, we examine the cleaning performance using constraints of varying accuracy. Specifically, we assess the accuracy of constraints by measuring an allowable *constraint filtering threshold* ρ for each applied constraint σ. During the selection of linear models for row constraints, we calculate the cumulative loss of each linear model across the entire dataset, ranked the models based on their losses, and then applied ρ as a criterion to limit the use of constraints. A higher ρ indicates lower accuracy in the constraints used for cleaning. We gradually relax ρ to get more coarse-grained constraints, and then report the change and comparison of the cleaning performance under constraints with different accuracy granularity.

The results are presented in Table 5. The L1-error of `MTSClean`-*soft* is not significantly affected by the accuracy of the constraints, whereas L1-error of `MTSClean` gradually increases as the constraint accuracy decreases. This aligns with the discussion in Section 3, as `MTSClean` consistently repairs erroneous data to the boundary of the constraint. `MTSClean`-*soft*, owing to the proposed superior cost function, balances the minimization principle with the degree of constraint violation when repairing erroneous data. This confirms the effectiveness of our proposed cost function.

As constraint precision decreases, the execution time of both methods slightly rises. This increase is due to a larger search space during optimization caused by reduced constraint reliability, necessitating longer computation time for repaired solutions. Additionally, the F1 score for both `MTSClean` and `MTSClean`-*soft* decreases with decreasing constraint accuracy. This decrease occurs because inaccurate constraints fail to identify certain subtle errors, leading to a reduction in error detection accuracy. Moreover, the *VRate* for both methods displays inconsistent variations, which can be attributed to potential degradation in constraint quality. This degradation may result in incorrect associations being identified, subsequently affecting repair outcomes. These findings highlight the suitability of our proposed repair cost function for time series data cleaning and emphasize the significance of constraint quality.

***Exp4*: The influence of various MVC strategies for repairing**. Considering that identifying genuinely erroneous data is the core aspect of data cleaning techniques, we conduct comparative experiments by replacing the violation-driven vertex cover strategy proposed in `MTSClean`-*soft* with several SOTA algorithms for the key cell determination on the constraint violation hypergraph. The results, presented in Table 6, demonstrate that our method achieves the best repair performance with the lowest time consumption. This underscores the effectiveness of our priority sorting strategy based violation degrees. As our method only requires a single prioritization function-based sorting, its complexity is comparable to the baseline `Shuffle`. Moreover, our approach identifies more accurate constraint sets, computes the MVC problem more efficiently, and results in a smaller final hypergraph size (*i.e.*, GSize). In contrast, methods like `MaxDegree`, `VertexSupport` have higher time complexities since they require recalculating the degrees of all vertices after each vertex removal. Specifically, `VertexSupport` calculates the support as the sum of the degrees of all neighbors of a vertex.

The experimental results validate that our heuristic priority function, based on constraint violation degrees, effectively improves the selection of critical cells for repair.

Table 6: Evaluation of various MVC strategies for repairing

Method	*L1*	*RAA*	*F1*	T(s)	VRate	GSize
MTSClean-Soft	**0.3297**	**0.812**	**0.973**	**82.31**	**0.394**	**0.1212**
Shuffle	0.7562	0.561	0.808	85.15	0.617	0.1238
MaxDegree	0.7431	0.560	0.809	181.97	0.474	0.1459
VertexSupport	0.7139	0.562	0.813	1029	0.582	0.2422
Greedy	0.7018	0.564	0.810	639.2	0.426	0.2151

6 RELATED WORK

Rule-based data cleaning. Relational data cleaning techniques mainly include rule-based [15, 29], statistical-based [3], human-in-the-loop [14, 34, 42], and learning-based [30] methods. Rule-based cleaning utilizes domain-specific knowledge and fully leverages the relationships in the data to improve data quality [1]. There are two types of repairs in this process: one is to fully trust the given constraints and only consider repairing the data; the other is to not fully trust the constraints, and to modify both the data and the (imperfect) constraints. The research on the former has been extensively conducted, which contains local cleaning (*e.g.*, [4]) and holistic cleaning. [5] proposed to detect conflicts in data representation based on hypergraph model, and developed cleaning tool Holoclean [20, 33]. [13] studied the multiple data cleaning on incompleteness and inconsistency with currency reasoning.

Time series data cleaning. Survey papers [11, 25, 41] summarized the research progress of time series data cleaning, mainly including smoothing-based, statistical-based, and constraint-based repair methods. [36] conducted data quality assessments on IoT data from the perspectives of data validity, integrity, and consistency. [6] formalized four types of data quality rules for temporal numerical data from the perspectives of “entity" and “attribute". [35] solved the time series data cleaning problem based on speed constraints on single sequence. In recent years, to support arithmetic operations, [24] provided conditional regression rules and discovered regression models from time series data. [16] proposed conformance constraints and evaluated them in trusted machine learning and data drift. Our recent work, TSDDiscover [8] mines accurate functional structure with allowable error bound to obtain expressive quality constraints from time series. Since the quality of constraints is of great importance for cleaning, the reliable mining of quality constraints for time series data also remains a direction worthy of further exploration.

7 CONCLUSIONS

This paper introduces a novel constraint-based data cleaning algorithm, `MTSClean`, capable of effectively repairing complex error instances in time series data. We further optimize its efficiency by proposing `MTSClean`-*soft*. A key innovation lies in utilizing the degree of constraint violation on the constraint violation hypergraph for rapid identification of genuinely erroneous cells. Additionally, we design a repair cost function tailored for time series data. Experimental results across multiple baseline algorithms confirm the effectiveness, efficiency, and necessity of our cleaning approach and strategy formulation. Our methods exhibit promise in enhancing the quality of time series data.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (2023YFB3308003); National Natural Science Foundation of China (NSFC) (62202126, 62232005, 92267203); China Postdoctoral Science Foundation (2022M720957); Heilongjiang Postdoctoral Financial Assistance (LBH-Z21137).

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDB J.* 24, 4 (2015), 557–581. https://doi.org/10.1007/S00778-015-0389-Y

[2] Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti, and Aditya P Mathur. 2017. WADI: a water distribution testbed for research in the design of secure cyber physical systems. In *Proceedings of the 3rd international workshop on cyber-physical systems for smart water networks*. 25–28.

[3] Asif Iqbal Baba, Manfred Jaeger, Hua Lu, Torben Bach Pedersen, Wei-Shinn Ku, and Xike Xie. 2016. Learning-Based Cleansing for Indoor RFID Data. In *SIGMOD*. ACM, 925–936. https://doi.org/10.1145/2882903.2882907

[4] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *SIGMOD*. ACM, 143–154.

[5] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou (Eds.). IEEE Computer Society, 458–469. https://doi.org/10.1109/ICDE.2013.6544847

[6] Tamraparni Dasu, Rong Duan, and Divesh Srivastava. 2016. Data Quality for Temporal Streams. *IEEE Data Eng. Bull.* 39, 2 (2016), 78–92. http://sites.computer.org/debull/A16june/p78.pdf

[7] Xiaoou Ding, Yingze Li, Chen Wang, Hongzhi Wang, and Haoxuan Li. 2023. Time Series Data Quality Rules Discovery with Both Row and ColumnDependencies. *Journal of Software (Chinese)* 34, 3 (2023), 1065–1086.

[8] Xiaoou Ding, Yingze Li, Hongzhi Wang, Chen Wang, Yida Liu, and Jianmin Wang. 2024. TSDDISCOVER: Discovering Data Dependency for Time Series Data. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 3668–3681. https://doi.org/10.1109/ICDE60146.2024.00282

[9] Xiaoou Ding, Yichen Song, Hongzhi Wang, Donghua Yang, and Yida Liu. 2023. Cleanits-MEDetect: Multiple Errors Detection for Time Series Data in Cleanits. In *28th International Conference, DASFAA*, Vol. 13946. 674–678.

[10] Xiaoou Ding, Yichen Song, Hongzhi Wang, Donghua Yang, Chen Wang, and Jianmin Wang. 2024. Clean4TSDB: A Data Cleaning Tool for Time Series Databases. *Proc. VLDB Endow.* 17, 12 (2024), 4377–4380. https://www.vldb.org/pvldb/vol17/p4377-wang.pdf

[11] Xiaoou Ding, Hongzhi Wang, Genglong Li, Haoxuan Li, Yingze Li, and Yida Liu. 2022. IoT data cleaning techniques: A survey. *Intell. Converged Networks* 3, 4 (2022), 325–339. https://doi.org/10.23919/ICN.2022.0026

[12] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Zijue Li, Jianzhong Li, and Hong Gao. 2019. Cleanits: A Data Cleaning System for Industrial Time Series. *Proc. VLDB Endow.* 12, 12 (2019), 1786–1789. https://doi.org/10.14778/3352063.3352066

[13] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Muxian Wang, Jianzhong Li, and Hong Gao. 2022. Leveraging Currency for Repairing Inconsistent and Incomplete Data. *IEEE Trans. Knowl. Data Eng.* 34, 3 (2022), 1288–1302. https://doi.org/10.1109/TKDE.2020.2992456

[14] Ju Fan and Guoliang Li. 2018. Human-in-the-loop Rule Learning for Data Integration. *IEEE Data Eng. Bull.* 41, 2 (2018), 104–115. http://sites.computer.org/debull/A18june/p104.pdf

[15] Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00439ED1V01Y201207DTM030

[16] Anna Fariha, Ashish Tiwari, Arjun Radhakrishna, Sumit Gulwani, and Alexandra Meliou. 2021. Conformance constraint discovery: Measuring trust in data-driven systems. In *Proceedings of the 2021 International Conference on Management of Data*. 499–512.

[17] Muhammad Fayaz, Shakeel Arshad, Abdul Salam Shah, and Asadullah Shah. 2018. Approximate methods for minimum vertex cover fail to provide optimal results on small graph instances: A review. *International Journal of Control and Automation* 11, 2 (2018), 135–150.

[18] Roland Fried and Ann Cathrice George. 2011. Exponential and Holt-Winters Smoothing. In *International Encyclopedia of Statistical Science*, Miodrag Lovric (Ed.). Springer, 488–490. https://doi.org/10.1007/978-3-642-04898-2_244

[19] Elad Hazan and Tomer Koren. 2012. Linear Regression with Limited Observation. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress. http://icml.cc/2012/papers/433.pdf

[20] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*. 829–846.

[21] Igor Ilic, Berk Görgülü, Mucahit Cevik, and Mustafa Gökçe Baydogan. 2021. Explainable boosted linear regression for time series forecasting. *Pattern Recognit.* 120 (2021), 108144.

[22] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning*. ACM Books, Vol. 28. ACM. https://doi.org/10.1145/3310205

[23] Ihab F. Ilyas and Theodoros Rekatsinas. 2022. Machine Learning and Data Cleaning: Which Serves the Other? *ACM J. Data Inf. Qual.* 14, 3 (2022), 13:1–13:11. https://doi.org/10.1145/3506712

[24] Rui Kang, Shaoxu Song, and Chaokun Wang. 2022. Conditional Regression Rules. In *ICDE*. IEEE, 2481–2493.

[25] Aimad Karkouch, Hajar Mousannif, Hassan Al Moatassime, and Thomas Noël. 2016. Data quality in internet of things: A state-of-the-art survey. *J. Netw. Comput. Appl.* 73 (2016), 57–81. https://doi.org/10.1016/J.JNCA.2016.08.002

[26] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks. In *ICDE*. IEEE, 13–24.

[27] Zijue Li, Xiaoou Ding, and Hongzhi Wang. 2020. An Effective Constraint-Based Anomaly Detection Approach on Multivariate Time Series. In *APWeb-WAIM*, Vol. 12318. 61–69.

[28] Zheng Liang, Hongzhi Wang, Xiaoou Ding, and Tianyu Mu. 2021. Industrial time series determinative anomaly detection based on constraint hypergraph. *Knowl. Based Syst.* 233 (2021), 107548.

[29] Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. 2020. Computing Optimal Repairs for Functional Dependencies. *ACM Trans. Database Syst.* 45, 1 (2020), 4:1–4:46. https://doi.org/10.1145/3360904

[30] Mohammad Mahdavi and Ziawasch Abedjan. 2021. Semi-Supervised Data Cleaning with Raha and Baran. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2021/papers/cidr2021_paper14.pdf

[31] Aditya P Mathur and Nils Ole Tippenhauer. 2016. SWaT: A water treatment testbed for research and training on ICS security. In *2016 international workshop on cyber-physical systems for smart water networks (CySWater)*. IEEE, 31–36.

[32] Zhongyi Pei, Zhiyao Cen, Yipeng Huang, Chen Wang, Lin Liu, Philip S. Yu, Mingsheng Long, and Jianmin Wang. 2024. BTTackler: A Diagnosis-based Framework for Efficient Deep Learning Hyperparameter Optimization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, Ricardo Baeza-Yates and Francesco Bonchi (Eds.). ACM, 2340–2351. https://doi.org/10.1145/3637528.3671933

[33] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[34] El Kindi Rezig, Mourad Ouzzani, Ahmed K. Elmagarmid, Walid G. Aref, and Michael Stonebraker. 2019. Towards an End-to-End Human-Centric Data Cleaning Framework. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019*. ACM, 1:1–1:7. https://doi.org/10.1145/3328519.3329133

[35] Shaoxu Song, Fei Gao, Aoqian Zhang, Jianmin Wang, and Philip S. Yu. 2021. Stream Data Cleaning under Speed and Acceleration Constraints. *ACM Trans. Database Syst.* 46, 3 (2021), 10:1–10:44. https://doi.org/10.1145/3465740

[36] Shaoxu Song and Aoqian Zhang. 2020. IoT Data Quality. In *CIKM*. ACM, 3517–3518. https://doi.org/10.1145/3340531.3412173

[37] Shaoxu Song, Aoqian Zhang, Jianmin Wang, and Philip S. Yu. 2015. SCREEN: Stream Data Cleaning under Speed Constraints. In *SIGMOD*. 827–841.

[38] Yunxiang Su, Yikun Gong, and Shaoxu Song. 2023. Time Series Data Validity. *Proc. ACM Manag. Data* 1, 1 (2023), 85:1–85:26. https://doi.org/10.1145/3588939

[39] Chen Wang, Xiangdong Huang, Jialin Qiao, Tian Jiang, Lei Rui, Jinrui Zhang, Rong Kang, Julian Feinauer, Kevin Mcgrail, Peng Wang, Diaohan Luo, Jun Yuan, Jianmin Wang, and Jiaguang Sun. 2020. Apache IoTDB: Time-series database for Internet of Things. *Proc. VLDB Endow.* 13, 12 (2020), 2901–2904.

[40] Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, and Jiaguang Sun. 2023. Apache IoTDB: A Time Series Database for IoT Applications. *Proc. ACM Manag. Data* 1, 2 (2023), 195:1–195:27.

[41] Xi Wang and Chen Wang. 2020. Time Series Data Cleaning: A Survey. *IEEE Access* 8 (2020), 1866–1881. https://doi.org/10.1109/ACCESS.2019.2962152

[42] Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. 2011. Guided data repair. *Proc. VLDB Endow.* 4, 5 (2011), 279–289. https://doi.org/10.14778/1952376.1952378

[43] Jing Nathan Yan, Oliver Schulte, Mohan Zhang, Jiannan Wang, and Reynold Cheng. 2020. SCODED: Statistical Constraint Oriented Data Error Detection. In *SIGMOD J, June 14-19, 2020*. ACM, 845–860.

[44] Aoqian Zhang, Shuqing Deng, Dongping Cui, Ye Yuan, and Guoren Wang. 2023. An Experimental Evaluation of Anomaly Detection in Time Series. *Proceedings of the VLDB Endowment* 17, 3 (2023), 483–496.

[45] Aoqian Zhang, Shaoxu Song, and Jianmin Wang. 2016. Sequential Data Cleaning: A Statistical Approach. In *SIGMOD*. 909–924.

[46] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S. Yu. 2017. Time Series Data Cleaning: From Anomaly Detection to Anomaly Repairing. *PVLDB* 10, 10 (2017), 1046–1057.