bNDCRepair: Cleaning both Data Errors and Inaccurate Constraints on Numerical Sequential Data

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

Muyun Zhou
Harbin Institute of Technology
22s003026@stu.hit.edu.cn

Yida Liu
Harbin Institute of Technology
22B903040@stu.hit.edu.cn

Chen Wang
Tsinghua University
chen_wang@tsinghua.edu.cn

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

Jianmin Wang
Tsinghua University
jimwang@tsinghua.edu.cn

ABSTRACT

Numerical sequence data from intelligent devices often have quality issues. While existing data cleaning methods focus on repairing data, we address the problem of repairing both data errors and inaccurate constraints. We propose two operations for modifying inaccurate constraints: expanding and compressing their value domains. Our solution includes constraint modification functions and algorithms to prevent under- and over-fitting in data cleaning. Theoretical evaluations demonstrate its reliability and effectiveness of the proposed solution, which achieves optimal repair with the distance no greater than $|\Sigma'_l| \cdot \epsilon_e + |\Sigma'_r| \cdot \epsilon_s$ from the optimal repair. Experiments on real-life and synthetic datasets show that our bNDCRepair method improves F1-score by 17.6% compared to using the original constraints and performs best in *MNAD*. Results show high-level performance with the combination of our bNDCRepair and the state-of-the-art CVtRepair and Clean4TSDB in sequential data tasks.

PVLDB Reference Format:
Xiaoou Ding, Muyun Zhou, Yida Liu, Chen Wang, Hongzhi Wang, and Jianmin Wang. bNDCRepair: Cleaning both Data Errors and Inaccurate Constraints on Numerical Sequential Data. PVLDB, 18(13): 5676 - 5688, 2025. doi:10.14778/3773731.3773742

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

1 INTRODUCTION

Numerical sequential data are defined as time-ordered sequences of values continuously measured and recorded at regular intervals [13, 23, 33]. High-quality data serves as the primary guarantee for reliable knowledge discovery, however, sequential data frequently suffer from quality issues, including noise, missing values, and inconsistencies [14, 15, 26]. To address this, data cleaning techniques are employed to identify and repair erroneous data, ensuring the generation of a valid, clean dataset, which is a critical preprocessing step for downstream analytics [9, 20]. A common strategy involves leveraging *data quality constraints*, either derived from domain expertise or extracted from business rules.

Table 1: Sensor data from an industrial draft fan unit.

Time	WSpeed	WDirect	Temp	Voltage	Electric	Power
t_1	10	270	20	100	30	3005
t_2	11	120	21	101	32	3202
t_3	9	140	25	99	31	3000
t_4	16	145	22	103	31	3150
t_5	13	-1	21	101	30	2999

Traditional constraint-based repair strategies rely exclusively on predefined data quality rules [20], with extensive research focusing on defining repair cost functions and optimizing for minimal modifications to dirty data [4]. However, this paradigm often provides suboptimal results, as constraints derived from expert knowledge or business processes may lack accuracy or contextual relevance for specific cleaning tasks. To address this limitation, recent approaches advocate for joint refinement of both data and constraints during repair. Formally framed as optimizing database instance I and constraint set Σ simultaneously to produce I' and Σ' such that I' satisfies Σ'. Notable advancements include Chiang's extension of FDs through attribute augmentation [5], Beskales' trust-aware repair algorithm incorporating data-constraint reliability weights [3], and Song's denial constraint-based framework for correcting oversimplified or overly restrictive rules [6, 35].

However, existing approaches overlook the distinct challenges of numerical sequential data, where coordinated refinement of both data errors and constraint inaccuracies is particularly critical, especially in IoT applications [23, 33]. This gap exacerbates two core issues: (*i*) ***Data dynamics***: Rapidly evolving IoT data distributions render static, expert-defined constraints obsolete, as prior domain knowledge may no longer align with current conditions, and (*ii*) ***Constraint mining limitations***: Constraints mined from noisy or small-scale IoT datasets inherit inherent biases (e.g., overfitting to sparse anomalies), leading to suboptimal repair guidance.

It is worth mentioning that the aforesaid repairing techniques prove inadequate for numerical data cleaning. Most existing constraint refinement techniques focus exclusively on predicate-level modifications (e.g., inserting or deleting predicates in one constraint) [5, 35], which are ill-suited for numerical domains for two critical reasons: (*i*) *Domain-specific limitations*: In industries with established knowledge bases (e.g., manufacturing), altering the number of predicates in a constraint is often prohibited, as such modifications may violate well-vetted operational protocols, and (*ii*) *Numerical precision requirements*: Many constraints (e.g., shown in Table 2) involve continuous value ranges that demand parameter

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 13 ISSN 2150-8097.
doi:10.14778/3773731.3773742

calibration rather than structural predicate adjustments. Below, we present a real case from IoT data quality management scenario.

Example 1.1. *Table 1 records sensor data from an industrial draft fan units in a power plant [12, 26], which suffers from both dirty data and inaccurate constraints. A fundamental physical principle dictates that power derives from voltage and current, formalized as a time series data dependency (TSDD) [8]:* $\forall t_i \in T, (t_i.Voltage, t_i.Electric \rightarrow^{f}_{(0.99,1.01)} t_i.Power, f = t_i.Voltage \cdot t_i.Electric)$. *However, real-world sensor fluctuations often violate the strict numerical bounds of such constraints, even when data remains physically valid. Direct predicate-level modifications are infeasible here, as this would violate the physical foundation of the rule. Similarly, for wind turbines, wind speed (WSpeed) and temperature collectively influence wind power (WP) through a parameterized constraint:* $k_1 \cdot WSpeed \cdot Temp \leq WP \leq k_2 \cdot WSpeed \cdot Temp$, *where* k_1, k_2 *are context-dependent coefficients. Existing constraint repair methods fail to adapt* k_1, k_2 *dynamically, as they lack mechanisms for numerical parameter calibration beyond rigid predicate manipulation.*

Example 1.1 highlight a critical gap: legacy approaches designed for categorical data or rigid logical rules are ill-suited for numerical sequential domains. Deploying inaccurate constraints in cleaning pipelines risks false positives (over-cleaning valid data) or false negatives (missing true errors), with severe operational consequences. While updating constraints during repair is essential, achieving mutually consistent clean data and clean constraints remains nontrivial. We face challenges including:
(*i*) ***How to repair & what to repair on both inaccurate constraints and data?*** When both the data and constraints exhibit inaccuracies, achieving high-quality repair necessitates a holistic integration of joint cost functions and coupled modification operations. It requires well-designed methods when considering the characteristic of numerical data.
(*ii*) ***How to avoid both over-fitting and under-fitting on repairing numerical data?*** Data-driven algorithms are prone to be over- or under-fitting when aiming to modify constraints from data. Both types of phenomena are expected to be resolved in the repairing process, especially for numerical data.
(*iii*) ***How to repair data with multiple (dependent) constraints?*** Multivariate numerical data often involves interdependent constraints, where modifications to one constraint may propagate inconsistencies to others. In such cases, it is easy to drop into an inconsistency constraint set unless well-consider the correlation among the modified objective.

Motivated by this, we solved an interesting data cleaning problem and proposed a novel modification on the value domain of constraints. Our **contributions** include:

(1). We formalize the "both repair data and constraints (bNDCRepair)" problem for the numerical sequential data which seriously suffer both dirty data values and inaccurate quality constraints. We propose two novel modification operations on constraints *w.r.t* numerical domains, respectively value range expansion and compression.

(2). We solve the bNDCRepair problem and well-consider under- and over-fitting issues in repair process. The proposed algorithm determines repair solutions of dependent constraints together to avoid conflicts and local optimality. We theoretically analyze kinds of theoretical results including the terminability and time efficiency of the proposed algorithm. We prove that the proposed method achieves optimal repair with the distance no greater than $|\Sigma_l'| \cdot \epsilon_e + |\Sigma_r'| \cdot \epsilon_s$ from the optimal repair, where Σ_l', Σ_r' are constraint sets and ϵ_s, ϵ_e are step sizes in the optimal repair searching.

(3). Through experiments on real-world datasets against five baseline methods, our bNDCRepair achieves 9.6% – 17.6% higher F1-scores in cleaning through constraint set refinement. By integrating bNDCRepair's constraint repair operations into CVtolerant which enables constraint predicates addition and deletion, we enhance repair quality, particularly for data with strong inter-attribute dependencies. By combining bNDCRepair with a state-of-the-art sequential data cleaner Clean4TSDB, we achieve high cleaning efficacy while maintaining competitive runtime efficiency.

Organization. Section 2 discusses the related work, and Section 3 formalizes the joint repairing problem. Section 4 details the violation detection phase, and Section 5 introduces constraint modification functions with the proposed bNDCRepair algorithm. Section 6 provides theoretical analysis. Experimental evaluations are presented in Section 7, and conclusions are drawn in Section 8.

2 RELATED WORK

Rule-based relational data cleaning mainly include rule-based [17, 27], statistical-based [2], human-in-the-loop [16, 31, 39], and learning-based [28] 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 data and the (imperfect) constraints. The research on the former has been extensively conducted, which contains local cleaning (*e.g.,* [4]) and holistic cleaning. [30] detected conflicts in data representation based on hypergraph model, and developed HoloClean. UniClean is a recent data cleaning solution proposed for mixed errors based on unified cleaners and optimized cleaning workflow [9, 10]. For the later, [5] proposed modifying both data and rules using the minimum description length principle and a unified repair model. [3] refined this by introducing confidence levels. [36] proposed using classifiers to decide repair targets. [29] minimized data modification costs within constraint thresholds to avoid over/under-fitting.

Numerical data cleaning. Quality issues in numerical data, especially IoT time series, have prompted focused research on specialized cleaning techniques. Surveys [13, 23] summarize progress in time series cleaning, covering smoothing methods such as EWMA [18], statistical methods such as Median [38], and constraint-based methods such as Speed (based on speed constraints (SC)) [34]. Tutorials [33] assess IoT data quality based on validity, integrity, and consistency. [7] formalizes data quality rules for temporal numerical data, which can be divided into constraints based on columns (i.e., the same sequence in the temporal context) and constraints based on rows (different attributes of the same tuple). Techniques now address complex quality requirements, like arithmetic operations and relaxed equality [19, 22, 24, 32, 40] for numerical data.

Our prior work includes TSDDiscover [8], which mines accurate functional structures with tolerance for error bounds to derive expressive quality constraints from time series, and Clean4TSDB

Table 2: Examples of data quality constraints compatible with DQR.

Data quality constraints	Semantic definition	Convertto DQR	Statement
tFD [21]	$c_{tFD} : X \rightarrow \Gamma(Y)$, $\Gamma = \{Y \in [l, u]\}$	$(c_{tFD}, [l, u], \{S[t_j].Y - S[t_i].Y \mid S[t_j].X = S[t_i].X\})$	When the attributes X of any two sequences are the same, the difference in attributes Y is within $[l, u]$.
SC [32]	$c_{SC} : sp_{min} \leq \frac{x_j - x_i}{t_j - t_i} \leq sp_{max}$	$(c_{SC}, [sp_{min}, sp_{max}], \frac{S[t_j] - S[t_i]}{t_j - t_i})$	The rate of change of the value of the attribute X in the Sequence is in the range $[l, u]$.
TSDD [8]	$c_{TSDD} : \forall t_i \in T$, $(t_a.X \rightarrow^{f}_{(l,u)} t_i.Y, f)$	$(c_{TSDD}, [l, u], \frac{f(S[t_i].Y)}{S[t_a].X})$	The relationship between the attributes X and Y of any two sequences satisfies $f(S[t_i].Y) - S[t_a].X$.
CRR [22]	$c_{CRR} : (f, \rho, C)$	$(c_{TSDD}, [-\rho, \rho], \{f(S[t].Y) - S[t].Y \mid t \in C\})$	If the timestamp t of the sequence belongs to C, then its attributes X and Y satisfy the relation $f(S[t].Y) - S[t].X$.

[11, 12], a rule-based cleaning method that effectively applies expressive constraints to repair errors in sequential data.

Building on our extensive experience in sequential data cleaning and dependency mining, we recognize the critical importance of refining imprecise constraints for high-quality repair. This paper complements existing methods by addressing the unique challenges of numerical data cleaning through constraint modification strategies, enabling more robust and context-aware repair outcomes.

3 PROBLEM OVERVIEW

3.1 Preliminaries

$I = \{S_1, ..., S_M\}$ denotes M-dimensional numerical sequential data instance, where $S = \langle s_1, ..., s_N \rangle$ is a sequence on I. $s_n = \langle x_n, t_n \rangle$, $(n \in [1, N])$, where x_n is a real-valued number with a time point t_n, $T = \{t_1, ..., t_n\}$ is the set of time points of I. Below we use $S[t]$ as the value of sequence S at time t for short. Σ denotes the set of constraints defined on data I, where $c \in \Sigma$ is a formulated or learnt constraint the data are expected to satisfy.

Given the diversity of quality rule formulations for sequential data (e.g., CRR and TSDD in Table 2), we first formalize the data quality requirements to establish a unified foundation for developing repair algorithms compatible with a consistent format.

Definition 3.1. ***(Data quality requirement)***. *Given I and $c \in \Sigma$, the triple* DQR*: $(c, [d_{\min}, d_{\max}], E)$ describe the data quality requirement, where c represents the identifier of the* DQR*, $d_{\min}$ and $d_{\max}$ are two numerical values that define a range. E is a function expression that specifies the condition to be satisfied by the data instance.* DQR *presents that the sequence set $S_X \subseteq I$ are regulated by the function E of constraints c, whose acceptable value domain locates in $[d_{\min}, d_{\max}]$. $S \in S_X$ is identified to* violate *c if S does not satisfy the content described by c, denoted by $S \not\models c$.*

Accordingly, existing data quality rules can be unified into the DQR format. Table 2 presents several commonly used data quality rules for numerical data. For instance, the TSDD introduced in Example 1.1 translated to the following DQR: $(c_1, [0.99, 1.01], \frac{S[t_i].\text{Power}}{S[t_i].\text{Electric} \cdot S[t_i].\text{Voltage}})$. We note that DQR could be well applied to numerical data for at least two reasons. On the one hand, DQR supports not only foundation relationships, *e.g.*, =, >, <, but also *arithmetic operations*. On the other hand, DQR supports the *relaxation* of the "equals" relationship. Therefore, DQR exhibits excellent scalability, allowing compatibility with newly proposed data quality rules in the future, making it more suitable for numerical data quality management [23, 37].

3.2 Problem statement

One crucial thing in bNDCRepair problem is to well-define the optimization objective, *i.e.*, the combined cost function *w.r.t* data and constraints. We apply the general cost function for data repairing in Definition 3.2 which is generally used in data quality techniques [20]. Since that the original Σ is considered to be inaccurate in the studied problem, we formalize the cost function of constraint repairing in Definition 3.3. Here, we propose two kinds of changes for DQR, *i.e.,* constraint expansion and compression, as discussed in detail in Section 5.2.

Definition 3.2. ***(Data repair cost)***. *Given the original data $S[t]$, let $S[t]'$ be the modified value of $S[t]$ according to the constraint set Σ, the repair cost between $S[t]$ and $S[t]'$ is $cost_d(S[t], S[t]'|\Sigma) = \frac{S[t] - S[t]'}{S[t]'}$, thus, the cost of modifying data I to I' with Σ is*

$$cost_d(I, I'|\Sigma) = \sum_{S \in I} \sum_{t \in T} cost_d(S[t], S[t]'|\Sigma) = \frac{S[t] - S[t]'}{S[t]'}.$$

Definition 3.3. ***(Constraint repair cost)***. *Let* DQR *$(c, [d_{\min}, d_{\max}], E)$ be modified to $(c', [d'_{\min}, d'_{\max}], E)$, such change comes from either the* expansion *or* compression *of the value domain of c* w.r.t *E. Let $[d^{\star}_{\min}, d^{\star}_{\max}]$ and $[d^{\bullet}_{\min}, d^{\bullet}_{\max}]$ denote one expansion and one compression, respectively. The cost of constraint repairing is*

$$cost_c(c, c') = \frac{|c'.d^{\star}_{\min} - c.d_{\min}| + |c'.d^{\star}_{\max} - c.d_{\max}|}{(c.d_{\max} - c.d_{\min}) + (c.'d^{\star}_{\max} - c.'d^{\star}_{\min})}$$
$$+\mu \frac{|c.'d^{\bullet}_{\min} - c.'d^{\star}_{\min}| + |c.'d^{\bullet}_{\max} - c.'d^{\star}_{\max}|}{(c'.d^{\star}_{\max} - c'.d^{\star}_{\min}) + (c'.d^{\bullet}_{\max} - c'.d^{\bullet}_{\min})},$$

where μ is a parameter adjusting the weight of the compression cost.

Constraint tightening enhances DQR's violation detection sensitivity, whereas generalization (via expansion) reduces detected violations by relaxing constraints. To encourage constraint refinements that expose latent inconsistencies while discouraging overly permissive modifications, we formulate the loss function with $\mu < 0$.

Since the repair for numerical data and constraints share similar modification forms, *i.e.,* to modify the inaccurate *value domain* to an accurate one, we unify the combined repair cost and propose the problem in Definition 3.4.

Definition 3.4. *Let Σ be the original set of constraints* w.r.t *I.* S(I) *and* S(Σ) *denote the set of all possible repairs of I and Σ, respectively. U records all possible repair pairs (Σ', I') such that I' satisfies Σ', denoted by $I' \models \Sigma'$, $\Sigma' \in$* S(Σ)*, $I' \in$* S(I)*. The **bNDCRepair Problem** is to find a repair pair $(\Sigma', I') \in U$ such that*

$$(\Sigma', I') = \arg \min_{(\Sigma', I')} cost((\Sigma', I'), (\Sigma, I)),$$

where $cost((\Sigma', I'), (\Sigma, I)) = cost_c(\Sigma, \Sigma') + \lambda cost_d(I, I'|\Sigma')$.

Accordingly, the optimization objective of bNDCRepair problem is $\arg \min_{\Sigma'} \{cost_c(\Sigma, \Sigma') + \lambda cost_d(I, I'|\Sigma')\}$, where λ is an adjustment coefficient.

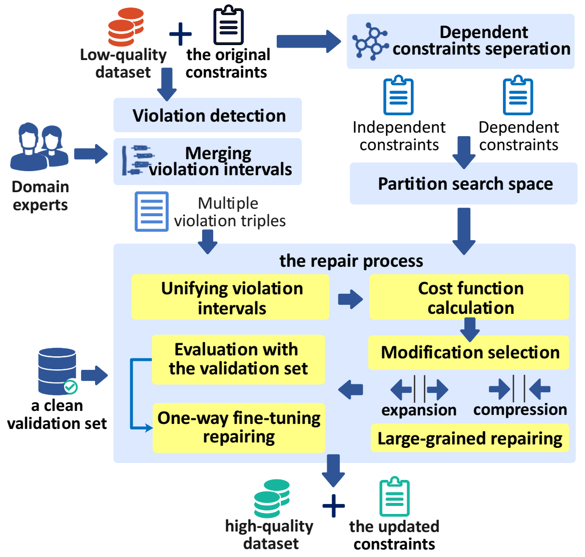


Figure 1: Overview of the bNDCRepair **solution.**

We note that the parameter λ is designed to balance the repair costs of constraints and data at comparable magnitudes [3] [35]. An excessively large λ biases the algorithm toward overreliance on raw data, leading to aggressive constraint modifications that risk underfitting (i.e., overly simplified constraints lose discriminative power). Conversely, an excessively small λ induces overconfidence in constraints, discouraging their refinement and allowing imprecise rules to propagate errors during cleaning. Theorem 6.1 formalizes a parameter estimation strategy for λ to mitigate these trade-offs.

3.3 Method overview

To address the bNDCRepair problem, we outline our solution in Fig.1. Based on the idea of minimizing the joint repair cost in Definition 3.4, we design a novel joint repair framework for constraints and data. The specific steps of the method are as follows:

(1). **Violation detection.** We first identify data errors and merges the time intervals for related errors to find errors that are consecutive in timestamps. Thus, we obtain a violation set *Vio* *w.r.t* errors and the involved constraints for the further steps.

(2). **Constraint set partition.** We identify each violated constraint as independent or dependent constraint by the intersection of the sequences contained in the expression, and consider the modification of dependent constraints together to achieve reliable repair of constraints.

(3). **Constraints and data repairing.** We propose an algorithm that jointly repairs data and constraints, which determines whether to expand or compress the value domain of constraints, as well as perform corresponding modifications to the data, based on the goal of minimizing the total repair cost of both constraints and data. Besides, we design one-way fine-tuning strategy with a small-scale clean validation dataset to guarantee the repairing effectiveness.

4 CONSTRAINT VIOLATION DETECTION

We first detect the error data in instance I *w.r.t* a given Σ including kinds of constraints, and obtain violations with the corresponding timestamps. To formalize, a violation set *Vio* contains all the violation units such as v: (S_i, c_k, t), which records the sequence value S_i violates constraint c_k at time t, denoted by $Vio = \{(S_i, c_k, t) | t \in T, S_i \not\models c_k \ w.r.t \ t\}$.

Note again that timestamp is always a built-in attribute for numerical sequence data, and dirty data occur in several consecutive time points, named as continuous errors [24]. Thus, we need to aggregate the *time intervals* of the detected violations in order to better identify continuous errors in numerical data. These violation intervals will further provide abundant evidence to the modification of either errors or inaccurate constraints. For two violation units $v_i(S, c, t_m)$ and $v_j(S, c, t_n)$ detected from sequence S *w.r.t* c with a given threshold δ, we merge t_m and t_n if $|t_m - t_n| < \delta$. Thus, we convert the violation unit $v(S, c, t)$ into a new form, *i.e.*, (S, c, T). We prefer to determine "$|t_m - t_n| < \delta$" with relaxation rather than "t_m and t_n strictly adjacent" because that not all continuous errors will occur in absolutely continuous time. Since that δ value really affects the merging result, the necessary manual participation is involved to determine a reliable δ value in our preprocess phase.

Algorithm 1: VioIntervalMerging

```
Input: Violation set $Vio$, distance threshold $\delta$
Output: $Vio$ after time interval merging
1 foreach $Vio(S,c) \subset Vio$ do
2     sort $(S, c, t)$s in $Vio(S, c)$ in ascending timestamp order;
3     modify all the $t$s in $(S, c, t)$ to $T_{t:t}$;
4     for $(S, c, T_{i:j}), (S, c, T_{k:l}) \in Vio(S, c)$ do
5         if $t_k - t_j < \delta$ then
6             $Vio(S, c) \leftarrow Vio(S, c) \setminus \{(S, c, T_{i:j}), (S, c, T_{k:l})\}$;
7             $Vio(S, c) \leftarrow Vio(S, c) \cup \{(S, c, T_{i:l})\}$;
8 return $Vio$
```

Algorithm 1 shows the merging process of violation units. We iteratively select the unit set $Vio(S, c)$ *w.r.t* S and c from Vio, and sort the elements in $Vio(S, c)$. Let $T_{i:j}$ denote the interval between t_i and t_j. We obtain interval $T_{m:n}$ if t_m and t_n of two adjacent elements satisfy $|t_m - t_n| < \delta$. Violation intervals will further provide abundant evidence for the follow-up repairing process.

Example 4.1. *Given three rules applied to the data in Table 1, i.e., one TSDD:* $\forall t_i \in T, (t_i.Voltage, t_i.Electric \xrightarrow{f}_{(0.99,1.01)} t_i.Power, f = t_i.Voltage \cdot t_i.Electric)$, *and two range constraints:* $20 \le Temp \le 21$, $9 \le WSpeed \le 11$, *we first transform them into* DQRs. *For convenience, we number the sequences in Table 1, with* S_1 *corresponding to WSpeed,* S_2 *to Temp,* S_3 *to Voltage,* S_4 *to Electric, and* S_5 *to Power. The converted* DQRs *are as follows:* $(c_1, [0.99, 1.01], \frac{S_5}{S_3+S_4})$, $(c_2, [20, 21], S_2)$ *and* $(c_3, [9, 11], S_1)$. *We obtain violations* $Vio = \{(\{S_3, S_4, S_5\}, c_1, t_4), (\{S_3, S_4, S_5\}, c_1, t_5), (S_2, c_2, t_1), (S_2, c_2, t_3), (S_2, c_2, t_4)\}, (S_1, c_3, t_4), (S_1, c_3, t_5)\}$ *from Table 1 after detection phase. Let* $\delta = 2$, *Algorithm 1 traverses the violation triples with the same sequence and the same constraint in* Vio. *For* S_2 *and* c_2, *it has* $t_3 - t_1 \le \delta$, *thus, we merge* $[t_1, t_3]$. *Finally, we obtain the merged set* $Vio = \{(\{S_3, S_4, S_5\}, c_1, t_{4:5}), (S_2, c_2, t_{1:4}), (S_1, c_3, t_{4:5})\}$.

5 REPAIRING BOTH DATA AND CONSTRAINTS

In this section, we first provide modification operations of constraints in Section 5.1, and discuss the concept of independent and dependent constraints in Section 5.2. Section 5.3 reports the repairing process, including the identification of independent versus dependent constraints and the specific ways to repair both data and inaccurate constraints.

5.1 Modification operation on constraints

For repairing inaccurate DQRs, we face with two kinds of modification of the value domain $[d_{\min}, d_{\max}]$ *w.r.t* one DQR, that is, *expansion* and *compression*. To achieve repair flexibility, we propose a two-stage modification mechanism for updating DQRs. We first compute to either expand or compress the range $[d_{\min}, d_{\max}]$ for each inaccurate DQR according to the optimization objective $cost((\Sigma', I'), (\Sigma, I))$. In this stage, both $d_{\min}$ and $d_{\max}$ are allowed to be modified, which is identified as double-boundary modification operation. In subsequent steps, we perform one-way fine-tuning operations for DQRs according to a small-scale validation dataset to improve the accuracy of the modified results.

5.1.1 Double-boundary modification operation. Definition 5.1 introduces the double-boundary modification operations, which achieves to satisfy the *local optimum* of the bNDCRepair problem.

Definition 5.1. *Given one* DQR *for data I, let* c.domain= $[d_{\min}, d_{\max}]$*, the* ***constraint compression*** *of c is to obtain another value domain* $[d'_{\min}, d'_{\max}]$ *by solving the optimization problem in Equation (1).*

$$\min\{cost_c(c, c') + \lambda cost_d(I, I'|c')\} \quad s.t. \quad d_{\max} \geq d'_{\max}, d_{\min} \leq d'_{\min} \tag{1}$$

Similarly, the ***constraint expansion*** *of* c.domain *is to solve the optimization problem in Equation (2).*

$$\min\{cost_c(c, c') + \lambda cost_d(I, I'|c')\} \quad s.t. \quad d_{\max} \leq d'_{\max}, d_{\min} \geq d'_{\min} \tag{2}$$

Algorithm 2: ConstraintCompress

```
Input: one DQR: $(c, [d_{\min}, d_{\max}], E)$ to be repaired, time interval $T_{\text{vio}}$
Output: the DQR after repair
1  $d'_{\min} \leftarrow d_{\min}, d'_{\max} \leftarrow d_{\max}, Mincost \leftarrow \lambda cost_d(I, I'|c)$
2  foreach $t \in T_{\text{vio}}$ do
3      if $S[t] - d'_{\min} \geq d'_{\max} - S[t]$ then
4          if the cost of changing $d'_{\min}$ to $S[t]$ is less than Mincost then
5              $Mincost \leftarrow cost_c([d'_{\min}, d'_{\max}], [S[t], d'_{\max}]) + \lambda cost_d(I, I'|[S[t], d'_{\max}])$;
6              change c.domain to $[S[t], d'_{\max}]$;
7      else
8          if the cost of changing $d'_{\max}$ to $S[t]$ is less than Mincost then
9              $Mincost \leftarrow cost_c([d'_{\min}, d'_{\max}], [d'_{\min}, S[t]]) + \lambda cost_d(I, I'|[d'_{\min}, S[t]])$;
10             change c.domain to $[d'_{\min}, S[t]]$;
11 return the updated DQR
```

We outline the constraint compression function in Algorithm 2. We first compute the cost of repairing only the data without modifying the constraint, and use it as the initial minimal repair cost. Therefore, we begin by calculating the cost of repairing out-of-bound data to the nearest boundary of the constraint and back up the initial constraint (line 1). Next, we iterate over the violation timestamps within the time interval T_{vio}. For each timestamp t, we compute the cost of jointly modifying the constraint and the data under the assumption that the current timestamp is trusted (line 3). If this cost is lower than the current minimal cost, we update the minimal cost and store the corresponding constraint (lines 4-10). After the iteration is complete, the resulting constraint is the repaired version that minimizes the total cost of modifying both the constraint and the dataset. Constraint expansion function could be performed in the similar way with Algorithm 2.

Example 5.1. *For the* DQR*: $(c_3, [9, 11], S_1)$ in Example 4.1, we first detect the violation locates in t_4 and t_5 from Table 1. Due to the small size of the example dataset, we use a relatively large λ to better illustrate how the algorithm modifies the constraint, e.g., $\lambda = 0.9$, $\mu = -1$. We first consider repairing the data using the* DQR*, i.e.,, modifying both $S_1[t_4]$ and $S_1[t_5]$ to 11, and treat this as the initial minimal cost. According to Definition 3.4, the repair cost is* $cost_c([9, 11], [9, 11]) + \lambda cost_d(I, I' \mid c) = 0 + 0.9 \cdot (\frac{16-11}{11} + \frac{13-11}{11}) =$ 0.56*. We then iterate over the violated timestamps to evaluate the total cost when trusting the timestamp and modifying the constraint. If the total cost is lower than the current minimal cost, we update the constraint accordingly.*

For t_4, since its value 16 is closer to the upper bound of $[9, 11]$*, we expand the upper bound to 16, resulting in a new interval* $[9, 16]$*. According to Definition 3.3, the cost of modifying the constraint is:* $cost_c([9, 11], [9, 16]) = \frac{|9-9|+|11-16|}{(11-9)+(16-9)} = 0.55$*. At this point, no data violates* $[9, 16]$*, so the data repair cost is* 0*. According to Definition 3.4, the total repair cost is* 0.55*. Since* $0.55 < 0.56$*, we update the current minimal cost to* 0.55 *and record the current best constraint as* $[9, 16]$*. Similarly, for t_5, we attempt to expand the constraint from* $[9, 11]$ *to* $[9, 13]$*. The cost of modifying the constraint, based on Definition 3.3, is 0.33. In this case, t_4 now violates the constraint and must be repaired to 13. The corresponding data repair according to Definition 3.2 cost is:* $(\frac{16-13}{13}) = 0.15$*. The total cost is* $0.33 + 0.9 \cdot 0.15 = 0.47$*, which is lower than the current minimum cost of* 0.55*. Therefore, we update the minimal cost to 0.47 and record the best constraint as* $[9, 13]$*. After iterating through all timestamps, we obtain the repair plan with the lowest total cost, which is to modify the constraint to* $[9, 13]$.

Though the double-boundary modification functions contribute to repairing imprecise DQRs, we notice that it is not sufficient to achieve high-quality repair only with such functions. There are two reasons for this. (*i*) This approach trends to be affected by the input dataset when modifying *c.domain*s, which could easily lead to over-fitting, and (*ii*) since we aim to repair all the inaccurate constraints, adjusting one *c.domain greedily* to an *optimal* interval does not necessarily result in the entire constraint set achieving the *optimal* goal. Therefore, fine-tuning the value ranges is necessary. By introducing a small-scale and clean validation dataset, we further propose one-way fine-tuning modification operations for DQRs.

5.1.2 One-way fine-tuning operation. Definition 5.2 proposes the fine-tuning operations, where we only modify *one bound* of *c.domain* with one meta-operation. The one-way fine-tuning operations essentially adopt the idea of discretizing continuous *c.domain*s. Both ϵ_e and ϵ_s represent the discretization *step size* calculated according to data distribution, and during each iteration, *c.domain* will be expanded or compressed in the unit of either ϵ_e or ϵ_s.

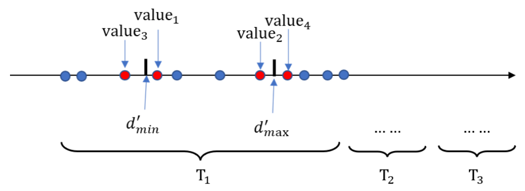


Figure 2: Location of the four values for fine-tuning operation.

Definition 5.2. *(**Compression meta-operation** ϵ_s). Given* c.domain $= [d_{\min}, d_{\max}]$ *defined on data I, when the left bound of* c.domain *is determined to be compressed, it has* $[d_{\min} + \epsilon_s, d_{\max}]$. *While it has* $[d_{\min}, d_{\max} - \epsilon_s]$ *with the right bound value to be updated.*

*(**Expansion meta-operation** ϵ_e).* c.domain *becomes* $[d_{\min} - \epsilon_e, d_{\max}]$ *when the left bound of is expanded, or it becomes* $[d_{\min}, d_{\max} + \epsilon_e]$ *with the updated right bound value.*

Since that the values of ϵ_e and ϵ_s are not easy to be determined, we propose modification strategy for ϵ_e and ϵ_s by satisfying the following two properties.

(1) During each change, meta-operation could not cross more than two data points, in case that it will not miss the minimum $cost_c$. To formalize, let $P(x, y)$ represent the set of data points in the time interval $T[x, y]$, we set

$$\epsilon_e = \arg\min\{|P(d_{\min} - \epsilon_e, d_{\min})|, |P(d_{\max}, d_{\max} + \epsilon_e)|\},$$

$$\epsilon_s = \arg\min\{|P(d_{\min}, d_{\min} + \epsilon_s)|, |P(d_{\max} - \epsilon_s, d_{\max})|\}.$$

(2) To deduce under-fitting, the updated *c.domain*s are supposed to cover more data points after several one-way fine-tuning changes. To formalize, we aim to find the changes satisfying $\frac{|P(d_{\min}-\delta_e, d_{\max})|}{|P(d_{\min}, d_{\max})|} > 1$, $\frac{|P(d_{\min}, d_{\max}+\delta_e)|}{|P(d_{\min}, d_{\max})|} > 1$, $\frac{|P(d_{\min}+\delta_s, d_{\max})|}{|P(d_{\min}, d_{\max})|} < 1$, and $\frac{|P(d_{\min}, d_{\max}-\delta_s)|}{|P(d_{\min}, d_{\max})|} < 1$.

The general idea to achieve the above purpose is to examine data point values near the boundaries, compute the distance between the left (resp. the right) bound and the data point, and select the *closest* one. Such distance is applied to determine the smallest unit for compression and expansion. We employ the method of *equal probability random sampling* to estimate ϵ_s and ϵ_e. We first get three time intervals T_1, T_2, T_3 by sampling in the entire time interval using uniformly distributed random numbers, which T_1, T_2 and T_3 are independent from each other. We estimate the values of ϵ_s and ϵ_e for each T, and calculate the average values derived from T_1, T_2 and T_3 to improve the accuracy of estimation. Specifically, we find four values in T_1, namely the minimum and maximum value value$_1^{T_1}$, value$_2^{T_1}$ satisfying $[d'_{\min}, d'_{\max}]$, the max value value$_3^{T_1}$ less than $d'_{\min}$ and the min value value$'_4$ greater than $d'_{\max}$. Fig. 2 presents the location of these four kinds of data points. Further, we are able to compute ϵ_e according to Equation (3).

$$\epsilon_e = \frac{1}{3}\sum_{i=1}^{3} \min\{|d'_{\min} - \text{value}_1^{T_i}|, |d'_{\min} - \text{value}_3^{T_i}|, |d'_{\max} - \text{value}_2^{T_i}|, |d'_{\max} - \text{value}_4^{T_i}|\} \tag{3}$$

ϵ_s can be computed analogously. To take the compression process for instance, the one-way fine-tuning function is presented in Algorithm 3. Line 3 starts to traverse all time points in T_1, T_2 and T_3, and identifies the position relationship between the current data and the boundary. After that, ϵ_s is calculated. According to the distance between the boundaries and the parameter values, it decides which bound in *c.domain* to be updated. Since Algorithm 3 needs to traverse three time intervals, and the operations of the loop could be completed in constant time, the time complexity of Algorithm 3 is $O(|T_1| + |T_2| + |T_3|)$.

Algorithm 3: ConstraintCompressS

```
Input: one DQR: $(c, [d_{\min}, d_{\max}], E)$, time intervals $T_1, T_2, T_3$
Output: the DQR after repair
1  $d'_{\min} \leftarrow d_{\min}, d'_{\max} \leftarrow d_{\max}$;
2  value$_1 \leftarrow d'_{\max}$, value$_2 \leftarrow d'_{\min}$, value$_3 \leftarrow d'_{\max}$, value$_4 \leftarrow d'_{\min}$;
3  for $T_i \in \{T_1, T_2, T_3\}$ do
4      foreach $t \in T_i$ do
5          compute value$_1$, value$_2$, value$_3$, value$_4$;
6      compute $\epsilon_s$ according to Equation (3);
7  if $|d'_{\min}$ − value$_1|$ or $|d'_{\min}$ − value$_3|$ is the smallest of these four
     values then
8      $d'_{\min} \leftarrow d'_{\min} + \epsilon_s$;
9  else
10     $d'_{\max} \leftarrow d'_{\max} - \epsilon_s$;
11 return $(c, [d'_{\min}, d'_{\max}], E)$
```

5.2 Dependent constraints partition

Since that multiple DQRs would be involved when error occurs, and such DQRs could not be treated independently in repairing process. In order to ensure the repairing reliability, we identify and cluster the independent DQRs as formalized in Definition 5.3.

Definition 5.3. *A* DQR: $(c, [d_{\min}, d_{\max}], E)$ *is regarded as an **independent** one if the sequences contained in E are disjoint with the sequences contained in any other* DQR. *Thus, a set of independent* DQR*s is denoted by* $\Sigma_{ind} = \{\text{DQR} | \bigcap_{i=1}^{n} E_i.S_X \neq \emptyset\}$. *Otherwise, the* DQR *is regarded as a **dependent** one.*

Obviously, the repair of inaccurate constraints in dependent DQRs are more challenged than the independent ones, for the consistency within multiple dependent inaccurate constraints needs to be taken into account in the repairing process. Accordingly, the search space for the optimization objective of bNDCRepair problem is further divided into search subspaces containing only *part of* constraints from Σ for different attributes. In the real cases that not much many constraints defined on the same sequence, the search subspace will not be quite large. Parallel searching strategies are performed on different subspaces in our solution, which significantly improve the search efficiency. To formalize, let $C(S_i)$ (resp. $C'(S_i)$) denotes the original (resp. updated) constraint set defined on sequence S_i, the optimization objective of sequence S_i in bNDCRepair problem is searched by $(\Sigma', I') = \arg\min\{cost_d(C(S_i), C'(S_i)) + \lambda cost_c(I, I'|C'(S_i))\}$.

5.3 The bNDCRepair algorithm

Figure 3 presents the constraints and data repairing process. We first perform an initial minimal repair on I using the constraint set and obtain the corresponding minimally repaired dataset I' along with its minimal repair cost. Then we execute the double-boundary modification operations, and involve validation dataset to determine whether there still exists violations *w.r.t* the updated

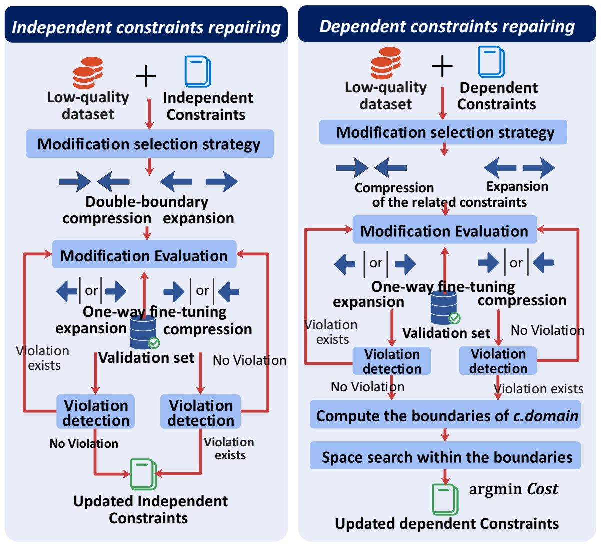


Figure 3: Repairing process for independent and dependent constraints

DQRs. If the repaired data violates the validation set, we continue to repair DQRs in the current Σ with one-way fine-tuning operations. During each iteration, $c.domain$ with the least cost in the search process is selected as the final result. We highlight the necessity of performing double-boundary modifications prior to the fine-tuning operations, because that if the one-way fine-tuning modifications are performed first, the algorithm cannot find the range of the optimal domain efficiently, and it is likely to suffer continuously searching the interval of the size of ϵ for much times.

Each time we attempt to modify a constraint, we simultaneously repair the data and compute the total repair cost from Definition 3.4. This total cost serves as the basis for deciding whether to repair the constraint, ensuring that the overall repair cost is minimized.

Since that it is much more complex to repair dependent constraints than independent ones, we first perform repair of independent constraints and data, followed by repairing dependent constraints and their associated data.

5.3.1 Independent constraints and data repair. We first find out the independent constraints by calculating the correlation degree of the constraints to determine whether there is a relationship between the constraints. Given two constraints c_i and c_j, let the violation interval sets corresponding to constraints c_i and c_j be $\{T_{i_{t_1,t_2}}, T_{i_{t_3,t_4}}, ..., T_{i_{t_{m-1},t_m}}\}$, $\{T_{j_{t_1,t_2}}, T_{j_{t_3,t_4}}, ..., T_{j_{t_{n-1},t_n}}\}$, respectively, thus, we quantify the relation degree between c_i and c_j with

$$rel(c_i, c_j) = \frac{2\sum_{x=1}^{m-1}\sum_{y=1}^{n-1}|T_{i_{t_x,t_{x+1}}} \cap T_{i_{t_y,t_{y+1}}}|}{\sum_{x=1}^{m-1}|T_{i_{t_x,t_{x+1}}}| + \sum_{y=1}^{n-1}|T_{i_{t_y,t_{y+1}}}|}.$$

For constraint c, we define a set $Rel(c) = \{c_i | rel(c, c_i) \geq confidence\}$, where confidence is determined as needed. The elements in $Rel(c)$ are constraints with strong correlation with c. Let $Mis(c)$ contain all the time points that violate any constraint in $Rel(c)$ but do not violate constraint c, that is, $Mis(c) = \{t | E[t] \models c_i, E[t] \not\models c, c_i \in Rel(c), \forall t\}$. Let c^s (resp. c^e) denotes the compressed (resp. expanded) constraint after the double-boundary modification phase, we compute the cost difference $\Delta cost(c, c^e, c^s)$. If $\Delta cost(c, c^e, c^s) \leq 0$, we choose to compress the constraint, because the cost of compression operation is smaller than expansion, and the number of constraints in $Mis(c)$ is not enough. Otherwise, we execute expansion functions $\Delta cost(c, c^e, c^s) = cost_c(c, c^s) + \lambda cost_d(I, I'|c^s) - cost_c(c, c^e) - \lambda cost_d(I, I'|c^e) - \omega|Mis(c)|$.

After we found the minimum cost repair for each constraint in Σ_{ind} under the current data set, we aim to solve the over-fitting issues with a validation dataset I_{valid} in with one-way fine-tuning functions. Here, I_{valid} is a verification dataset determined by experts, and it satisfies: 1) the schema in I_{valid} is the same as dataset I, but the data amount is much smaller than I, and 2) data in I_{valid} are absolutely clean. Although the optimal repair on I_{valid} is not necessarily the real optimal repair, it can be used as a reference to appropriately reduce the overfitting phenomenon of Σ'.

Algorithm 4: IndConstraintandDataUpdate

```
   Input: the instance $I$ to repair, the constraint set $\Sigma$ to repair
   Output: the udpated $\Sigma$, the updated $I$
1  for DQR: $(c_i, [d^i_{\min}, d^i_{\max}], E) \in \Sigma$ do
2      if $\Delta cost(c, c^e, c^s) \leq 0$ then
3          Update $c_i$ by ConstraintCompress;
4      else
5          Update $c_i$ by ConstraintExpand;
6      if $I_{\text{valid}}$ contains data violating $\Sigma'$ then
7          while $I_{\text{valid}}$ has data violating $\Sigma'$ do
8              Update $c'_i$ by ConstraintExpandS;
9      else
10         while No violation in $I_{\text{valid}}$ w.r.t $\Sigma'$ do
11             Update $c'_i$ by ConstraintCompressS;
12     Update DQR in $\Sigma$ and $I$;
13 return $\Sigma$, $I$
```

Algorithm 4 outlines the repair process of independent constraints and corresponding data. For each constraint, it first determines either to compress or expand the value domain according to the strategy mentioned above. Then it checks whether data in I_{valid} violate the current constraint, and it calls one-way fine-tuning expansion until no violation (lines 6-8). Otherwise, if there is no violation in I_{valid}, one-way compression will be perform to narrow the value domain until appearing violation (lines 9-11). After each constraint adjustment, we simultaneously retain the corresponding minimally repaired dataset I under the updated constraint. Finally, it chooses value domain with the lowest cost one as output. It is worth mentioning that the repair result is not related to the size of I_{valid}, but to the quality of I_{valid}.

Proposition 5.1. ***The time complexity of Algorithm 4 is*** $O(|\Sigma| \cdot |T| \cdot k \cdot (|T_1| + |T_2| + |T_3|))$***, where*** k ***is the upper bound of the execution times of the loop in lines 6-11.***

Proof. Since Algorithm 4 needs to traverse each constraint in the constraint set, during the traversal process, it is necessary to call the double-boundary modification functions. The "while loop" in it may or may not be triggered, so the time complexity in the best case is $O(|\Sigma| \cdot |T|)$. If the "while loop" is triggered, the time complexity is $O(|\Sigma| \cdot |T| \cdot k \cdot (|T_1| + |T_2| + |T_3|))$. And we further discuss k in Theorem 6.2. □

5.3.2 *Dependent constraints and data repair.*

For the update of dependent constraints, we have reduced the search spaces with the optimization objective mentioned in Section 5.2. As a result, we only need to consider the interaction between the constraints defined on the same sequence. Specifically, we propose **DConstraintUpdate** to repair the dependent constraints defined in the same sequence. **DConstraintUpdate** performs the constraint modification functions with $Mincost = cost_d(I, I'|\{c_1^s, ..., c_m^s\})$ instead of $cost_d(I, I'|c)$, with the other steps similar to Algorithm 4. In this way, inconsistencies and local optimality caused by repairing each constraint alone can be effectively avoided.

Since that some DQRs are defined on multiple sequences, such DQRs are likely to be divided into multiple constraint subsets when the repairing algorithm process each sequence, and the modification results provided by different subsets may not align. In order to ensure the consistency of constraint repairing, we determine the final modification result *e.g., c.domain* of one DQR from an alignment process. Suppose we obtain the updated value domains of constraint c_{updated} defined on l sequences, denoted by $[d_{\min}^{s_{i_1}}, d_{\max}^{s_{i_1}}]$, $[d_{\min}^{s_{i_2}}, d_{\max}^{s_{i_2}}]$, ..., $[d_{\min}^{s_{i_l}}, d_{\max}^{s_{i_l}}]$. We limit the minimum value on the left side of $c_{\text{updated}}.domain$ to $\min\{d_{\min}^{s_1}, ..., d_{\min}^{s_l}\}$ and the maximum value on the left side of the range to $\max\{d_{\min}^{s_1}, ..., d_{\min}^{s_l}\}$, respectively, for updating the DQR. Similarly, we limit the minimum value and maximum value on the right side of $c_{\text{updated}}.domain$ to $\min\{d_{\max}^{s_1}, ..., d_{\max}^{s_l}\}$ and $\max\{d_{\max}^{s_1}, ..., d_{\max}^{s_l}\}$. In this case, we obtain the aligned optimal domain of the updated DQR with the aforesaid modification strategies.

Algorithm 5 presents the process of repairing dependent constraints. We begin with violation detection on I, obtain the aggregated violation intervals, and then separate independent constraints. If there is no dependent constraints, we just need to repair Σ_{ind}s (lines 3-4). Otherwise, we repair independent constraints first, and then repair dependent constraints (lines 6-9). Here, the proposed modification functions are performed. After updating the constraints, we align the modification results of each constraint (lines 9-15). After each constraint adjustment, we retain the corresponding minimally repaired data I under the updated constraint. We then obtain the final updated constraint set Σ' and data I'.

Proposition 5.2. ***The time complexity of Algorithm 5 is*** $O(\max\{|\Sigma_{ind}|, |\Sigma\backslash\Sigma_{ind}|\} \cdot |T| \cdot (|T_1| + |T_2| + |T_3|))$.

Proof. If constraints in Σ are all independent constraints, Algorithm 5 costs $O(|\Sigma| \cdot |T| \cdot k \cdot (|T_1| + |T_2| + |T_3|))$ in time, If constraints are not all independent constraints and parallel operations are not considered, **DConstraintUpdate** in line 8 costs $O(|\Sigma_{ind}| \cdot |T| \cdot k \cdot (|T_1| + |T_2| + |T_3|))$, thus the time complexity of the "for loop" in lines 7-8 is $O(|\Sigma\backslash\Sigma_{ind}| \cdot |T| \cdot k \cdot (|T_1| + |T_2| + |T_3|))$, and the second "for loop" in lines 9-16 costs $O(| \cap_{S_i \in I} \Sigma_{S_i}| \cdot l \cdot (|T_1| + |T_2| + |T_3|))$, where l is a constant. Thus, Algorithm 5 costs $O(\max\{|\Sigma_{ind}|, |\Sigma\backslash\Sigma_{ind}|\} \cdot |T| \cdot k \cdot (|T_1| + |T_2| + |T_3|))$ in time. □

After the above process, we obtain the updated Σ' and the corresponding repaired dataset I' with the minimal total modification cost. We present the repairing process in Example 5.2 below.

```
Algorithm 5: ConstraintandDataRepair
   Input: the instance $I$ to repair, the constraint set $\Sigma$ to repair
   Output: the final repaired constraint set $\Sigma'$ and data $I'$
 1 Perform violation detection process discussed in Section 4;
 2 Separate independent dependent constraints set into $\Sigma_{ind}$ and $\Sigma_d$
     respectively;
 3 if $\Sigma_{ind} \cap \Sigma = \Sigma$ then
 4     $\Sigma', I' \leftarrow$ Algorithm 4$(I, \Sigma)$;
 5 else
 6     $\Sigma, I \leftarrow$ Algorithm 4$(I, \Sigma_{ind})$;
 7     for $S_i \in I$ do
 8         $\Sigma_{S_i} \leftarrow$ DConstraintUpdate$(I, S_i.\Sigma_d)$;
 9     for $c \in \cap_{S_i \in I} \Sigma_{s_i}$ do
10         Take the min and max values of all left-hand intervals of $c$
             as the search range of the left-hand interval, and the same
             as right-hand interval;
11         while Current interval is still within the limit do
12             if $\Delta cost(c, c^e, c^s)) \leq 0$ then
13                 Update $c_i'$ by ConstraintCompressS;
14             else
15                 Update $c_i'$ by ConstraintExpandS;
16         Select arg min $cost_c(I, I') + \lambda cost_d(I, I'|\Sigma')$ and update it
             to $\Sigma'$ and $I'$;
17     return $\Sigma', I'$
```

Example 5.2. *Recall* c_3 $9 \leq WSpeed \leq 11$ *and another two constraints* c_4 : $30 \leq Electric \leq 30$, *and one SC* c_5 : $1.7 \leq \frac{Temp}{WSpeed} \leq 2.4$, *the converted* DQR*s set is as follows:* $\Sigma = \{(c_3, [9, 11], S_1)$, $(c_4, [30, 30], S_4)$, $(c_5, [1.7, 2.4], \frac{S_2}{S_1})\}$, *we first separate independent constraints. Since* c_3 *and* c_5 *are defined on the same sequence* S_1, *they are identified as dependent constraints, while* c_4 *is defined on sequence* S_4, *so it is regarded as one independent constraint. We partition the search space for repairing, separating constraints for each sequence (e.g.,* c_3, c_5 *for* S_1*) and performing parallel repairs. Next, we repair inaccurate constraints by minimizing a combined cost function, updating constraint domains accordingly. For* S_1, *the optimal range for* c_3 *is [9,16] with repair costs of 0.56 for* c_3 *and 1.42 for* S_1. *This process is applied to both independent and dependent constraint sets, yielding search spaces as* $\{(c_3', [9, 16], S_1), (c_5', [1.375, 2.4], \frac{S_2}{S_1})\}$, $\{(c_5', [1.375, 2.8], \frac{S_2}{S_1})\}$, $\{(c_4', [30, 32], S_4)\}$.

We then algin the updated c.domains *from different search space for one constraint* c. *Note that* c_5 *is defined on two sequences, it has two outputs in the previous step, namely* $[1.375, 2.4]$ *and* $[1.375, 2.8]$. *So we need to unify the output to one value domain* $[d_{\min}^{\text{uni}}, d_{\max}^{\text{uni}}]$. *We restrict the left side of the final domain* $d_{\min}^{\text{uni}}$ *to no more than the left side of these two outputs, that is,* $1.375 \leq d_{\min}^{\text{uni}} \leq 1.375$. *Similarly, it has* $2.4 \leq d_{\max}^{\text{uni}} \leq 2.8$. *Accordingly, the one-way fine-tuning operations are preformed to search for the lowest cost value domain. Therefore, we obtain the updated constraint set* $\Sigma' = \{(c_3', [9, 16], S_1), (c_4', [30, 32], S_4), (c_5', [1.375, 2.5], \frac{S_2}{S_1})\}$. *and its corresponding repaired data* $S_1 = \{10, 11, 9, 16, 13\}$, $S_2 = \{24, 20, 22.5, 22, 21\}$ *and* $S_4 = \{30, 32, 31, 31, 30\}$.

6 THEORETICAL DISCUSSION

In this section, we analyze the reliability and effectiveness of our bNDCRepair solution. We also introduce two algorithm properties and formally discuss the repairing effectiveness.

We first focus on the parameter λ in the combined cost (Definition 3.4). When constraints and data are treated equally, we present the formula for computing λ in Theorem 6.1.

Theorem 6.1. ***Parameter estimation of*** λ. $\lambda = \max\left\{\frac{cost_c(\Sigma,\Sigma')}{cost_d(I,I'|\Sigma)-cost_d(I,I'|\Sigma')}, \max_{c\in\Sigma}\{-\frac{cost_c(c,c')}{cost_d(I,I'|c')}\}\right\}$.

Proof. According to the repair cost function, once the constraints are updated, the value of the objective function needs to be increased, *i.e.,* $cost_c(c,c') + \lambda \cdot cost_d(I,I'|c') \geq 0$, otherwise it violates the definition of repair cost. We obtain $\lambda \geq -\frac{cost_c(c,c')}{cost_d(I,I'|c')}$. Besides, the precondition of bNDCRepair is $\lambda \cdot cost_d(I,I'|\Sigma) > cost_c(\Sigma,\Sigma') + \lambda \cdot cost_d(I,I'|\Sigma')$, and $\lambda > \frac{cost_c(\Sigma,\Sigma')}{cost_d(I,I'|\Sigma)-cost_d(I,I'|\Sigma')}$. When performing constraint compression of each c in Σ, we also calculate the cost of data repairing under the updated c', thus $\max -\frac{cost_c(c,c')}{cost_d(I,I'|c')}$ is selected as the estimated value of λ. While, the estimated λ comes from $\frac{cost_c(\Sigma,\Sigma')}{cost_d(I,I'|\Sigma)-cost_d(I,I'|\Sigma')}$ when performing expansion on c. To formalize, λ is selected from the larger one, *i,e.,* $\lambda = \max\left\{\max_{c\in\Sigma}\{-\frac{cost_c(c,c')}{cost_d(I,I'|c')}\}, \frac{cost_c(\Sigma,\Sigma')}{cost_d(I,I'|\Sigma)-cost_d(I,I'|\Sigma')}\right\}$. □

Since there are two kind of basic operations for constraint repairing, *i.e.,* expansion and compression, it is important to ensure that the operation on one constraint do not occur alternately, *i.e.,* the constraint repairing process can converge. This prevents us from doing a lot of useless intermediate repairing. Theorem 6.2 and Theorem 6.3 guarantee the converge of the iterative modifications. It further show the potential of our algorithm to be scalable.

Theorem 6.2. ***(Terminability). The loops in repairing the constraint set*** Σ ***are guaranteed to terminate.***

Proof. We need to prove that the upper bound of the execution times of the statement in Algorithm 4 (line 7), *i.e,* k in Proposition 5.1 is computable. Given the finite cardinality of I_{valid}, let Up (resp. Lo) to be the max (resp. min) value in sequence S from I_{valid}, the updated *c.domain* will certainly cover the both bounds of I_{valid} after a limited number of either expansion or compression operations. Thus, k is determined by $\max\left\{\lceil\frac{(d_{\max}-Up)+(Lo-d_{\min})}{\delta_e}\rceil, \lceil\frac{(Up-d_{\max})+(d_{\min}-Lo)}{\delta_s}\rceil\right\}$, i.e., the execution times of algorithm has an upper bound, ensuring termination. □

Theorem 6.3. ***(Unidirectional). Algorithm 5 guarantees that the value domain updates in constraints are in the same direction for either compression or expansion, rather than alternating compression and expansion.***

Proof. The guarantee of unidirection in constraint repairing focuses on the loop in Algorithm 5 lines 11-15. Given $\Delta cost(c,c^e,c^s)$, only when $cost_c(c,c^s) + \lambda cost_d(I,I'|c^s) - cost_c(c,c^e) - \lambda cost_d(I,I'|c^e) > 0$ and $\Delta cost(c,c^e,c^s) \leq 0$, the constraint domain may fluctuate. In this case, the proposed algorithm uses the compression operation and obtain c^s_{update}. When the loop reaches this position again, if the algorithm chooses to expand the current constraint, it will return to c. According to the selection strategy, it has $\lambda cost_d(I,I'|c^s_{\mathsf{update}}) > -cost_c(c,c^s_{\mathsf{update}}) + \lambda cost_d(I,I'|c) + \omega|Mis(c^s)|$. This exactly shows that when the constraint are compressed to c^s_{update}, the modification cost of the dataset is too high. So the occurrence of fluctuation is reasonable. And at this time, the fluctuation only appears between such two values. To avoid fluctuation, we could simply add a judgment logic to the algorithm to detect whether there is a phenomenon of fluctuation between this two values, accordingly, Algorithm 5 computes to update the constraint domain in the same direction. □

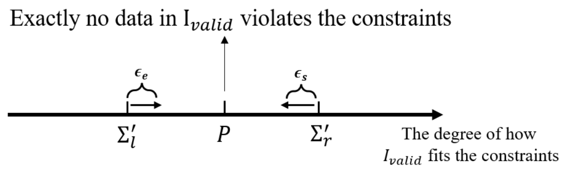

Exactly no data in I$_{valid}$ violates the constraints
ϵ_e
ϵ_s
Σ'_l
P
Σ'_r
The degree of how I_{valid} fits the constraints

Figure 4: The relationship between Σ'_l, Σ'_r and P.

We next evaluate the theoretical effectiveness of the proposed repair algorithm. With two parameters θ and γ, we propose the optimal repair of bNDCRepair problem in Definition 6.1, and Theorem 6.4 presents the upper bound of distance to the optimal solution.

Definition 6.1. *Given θ and γ, (I^*,Σ^*) is regarded as* ***the optimal repair of*** *bNDCRepair* ***problem*** *if it satisfies* $cost_d(I_{\mathsf{valid}}, I'_{\mathsf{valid}}|\Sigma^*) < \theta$ *and* $cost_c(\Sigma,\Sigma^*) < \gamma$ *and* $(I^*,\Sigma^*) = \arg\min cost_c(\Sigma,\Sigma') + \lambda cost_d(I,I'|\Sigma')$.

Intuitively, the optimal repair (I^*,Σ^*) should not only minimize the joint repair cost, but also ensure that the cost of modifying the validation set to satisfy Σ^* falls within a reasonable range. The restriction of $cost_d(I_{\mathsf{valid}}, I'_{\mathsf{valid}}|\Sigma^*) < \theta$ is because the data in the validation set is clean, and the repaired constraint set needs to conform to the data distribution of I_{valid}. While, *c.domain* is not expected to expand indefinitely to avoid under-fitting, thus the scope of constraint modification *i.e.,* $cost_c(\Sigma,\Sigma^*)$ is set to be not larger than a given γ.

Theorem 6.4. ***(Optimal solution gap). Given the updated constraint set*** Σ' ***from the proposed repairing algorithm, the gap between*** Σ' ***and the optimal repair*** Σ^* ***is no larger than*** $|\Sigma'_l| \cdot \epsilon_e + |\Sigma'_r| \cdot \epsilon_s$***. That is,***

$$\textit{Dist}(\Sigma',\Sigma^*) \leq |\Sigma'_l| \cdot \epsilon_e + |\Sigma'_r| \cdot \epsilon_s,$$

where $\textit{Dist}(\Sigma',\Sigma^*)$ ***denotes to the distance between the update constraint set*** Σ' ***and the optimal repair set*** Σ^****,*** ϵ_e ***(resp.***ϵ_s***) denotes the step size in expansion (resp. compression) meta-operation,*** $\Sigma'_l \subseteq \Sigma'$ ***denotes the constraints still violated by data in*** I_{valid}***, and*** $\Sigma'_r = \Sigma' \backslash \Sigma'_l$***.*** $|\Sigma'_l|$ ***(resp.***$|\Sigma'_r|$***) denotes the total number of constraints in*** Σ'_l ***(resp.***Σ'_r***).***

Proof. After obtaining the initial Σ' through double-boundary modification operations on Σ, we use I_{valid} to validate Σ'. The constraints in Σ' can then be divided into two subsets: the set of constraints satisfied by the data, denoted as Σ'_r, and the set of constraints violated by the data, denoted as Σ'_l. Assuming there exists a set of constraints P that perfectly satisfies the validation set I_{valid}, then the relationship between each constraint in Σ'_r and Σ'_l with respect to P is illustrated in Figure 4.

For example, assuming that the data in Table 1 corresponds to I_{valid}, then for the range constraint on the attribute Power, the DQR $\{c_1, [2999, 3202]\}$ is exactly at point P, the DQR $\{c_2, [2990, 3210]\}$ lies to the right of point P, and the DQR $\{c_3, [3000, 3200]\}$ lies to the left of point P.

We define θ as the cost of repairing I_{valid} using Σ', denoted as $\text{cost}_d(I_{\mathsf{valid}}, I'_{\mathsf{valid}} \mid \Sigma^*)$, and define γ as the cost of modifying the initial constraint set Σ to Σ', denoted as $\text{cost}_c(\Sigma,\Sigma')$. Under these

Table 3: Summary of the comparison cleaning methods.

Method	Introduction	Supported constraints	Considering modifying constraints?
oriRepair	a general constraint-based cleaning method	FD	
Holoclean [30]	a general SOTA cleaning method	DC	
Speed [34]	the speed constraint-based SOTA sequential cleaning method	SC	
CVtRepair [35]	a cleaning algorithm that supports constraint predicate deletion and insertion	DC	√
Clean4TSDB [11]	a recent constraint-based SOTA time series cleaning method	SC, TSDD	
bNDCRepair (Ours)	the cleaning algorithm used in this paper supports constraint range modification	TSDD	√
bNDC + CVtRepair	combines the proposed bNDCRepair and CVtRepair	TSDD	√
bNDC + Clean4TSDB	combines the proposed bNDCRepair and Clean4TSDB	TSDD	√

settings, the optimal repair Σ^* is the one within the range bounded by Σ'_l and Σ'_r that minimizes the total repair cost. Since the constraint domain is continuous, and the exactly optimal repair Σ^* is difficult to find, our solution discretizes the continuous constraint domain, and introduces ϵ_e and ϵ_s to denote the size of steps after constraint domain discretization, as shown in Section 5.1.2. For the constraints in Σ'_l *expansion* is performed to search in step of ϵ_e, while *compression* is used in the searching step of ϵ_s for the constraints in Σ'_r. When the total cost of joint repair reaches its minimum, the search process terminates, yielding the updated constraint set Σ'. In this situation, although Σ' may not exactly equal Σ^*, the distance between them is guaranteed to be within the range of $|\Sigma'_l| \cdot \epsilon_e + |\Sigma'_r| \cdot \epsilon_s$, otherwise, the *expansion* and *compression* steps would not have terminated. □

Besides the optimal solution gap *w.r.t* the whole constraint set Σ, we present the theoretical analysis of the upper bound for repairing *one* constraint in Theorem 6.5.

Theorem 6.5. ***(Theoretical upper bound for single constraint repairing). Given one inaccurate*** DQR $(c, [a, b], E)$ ***for sequence*** S***, let*** $f(x)$ ***denote the probability distribution function of*** S***, and*** $[a^*, b^*]$ ***denotes the optimal repair of the value range*** $[a, b]$***, it has*** $a^*, b^* = \arg\min_{a^*,b^*}\{\frac{|a-a^*|+|b-b^*|}{|b-a|+|b^*-a^*|} + \lambda H(a^*, b^*)\}$***, where*** $H(a^*, b^*) = \int_{\min(S)}^{a^*}(a^* - x)f(x)\,dx + \int_{b^*}^{\max(S)}(x - b^*)f(x)\,dx$***. The gap between the updated value range*** $[a', b']$ ***by Algorithm 5 and the optimal one*** $[a^*, b^*]$ ***is***

$$\textsc{Dist}([a^*, b^*], [a', b']) = |a^* - a'| + |b^* - b'| \le \epsilon_e$$

when I_{valid} ***violates***$[a', b']$***, and otherwise, it is no more than*** ϵ_s.

Proof. According to the constraint repair cost function in Definition 3.3, the cost of repairing $[a, b]$ to $[a^*, b^*]$ is $\frac{|a-a^*|+|b-b^*|}{|b-a|+|b^*-a^*|}$. As shown in Fig. 5, the cost of repairing data that smaller than a^* in S to a^* is $\int_{\min(S)}^{a^*}(a^* - x)f(x)\,dx$, while the cost of repairing data that larger than b^* to b^* is $\int_{b^*}^{\max(S)}(x - b^*)f(x)\,dx$. In this case, the sum cost of repairing S is

$$cost(S, S'|c^*) = \int_{\min(S)}^{a^*}(a^* - x)f(x)\,dx + \int_{b^*}^{\max(S)}(x - b^*)f(x)\,dx,$$

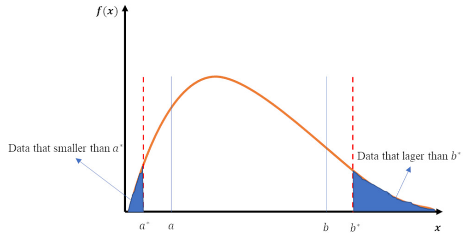


Figure 5: Data distribution and the constraint domain.

According to Equation (1), the combined repair cost *cost* is

$$\frac{|a - a^*| + |b - b^*|}{|b - a| + |b^* - a^*|} + \lambda\left(\int_{\min(S)}^{a^*}(a^* - x)f(x)\,dx + \int_{b^*}^{\max(S)}(x - b^*)f(x)\,dx\right)$$

The optimal repair of the given DQR is the solution that minimizes *cost*, so it has $a^*, b^* = \arg\min_{a_{\text{update}}, b_{\text{update}}} cost$.

Further, according to Theorem 6.4, if I_{valid} violates the given DQR, it has $|\Sigma'_l| = 1$ and $|\Sigma'_r| = 0$. Thus, the distance between the output repair $[a', b']$ and the optimal $[a^*, b^*]$ is less than $|\Sigma'_l| \cdot \epsilon_e + |\Sigma'_r| \cdot \epsilon_s = \epsilon_e$. Otherwise, $|\Sigma'_l| = 0$ and $|\Sigma'_r| = 1$. So the gap is less than $|\Sigma'_l| \cdot \epsilon_e + |\Sigma'_r| \cdot \epsilon_s = \epsilon_s$, which confirms Theorem 6.4. □

7 EXPERIMENTAL EVALUATION

7.1 Experimental setting

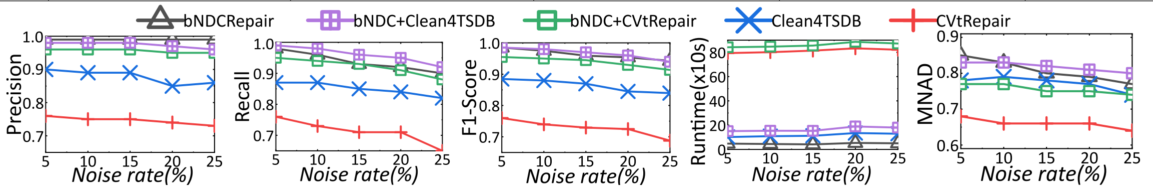
Experimental dataset. We use four real-world datasets and a synthetic dataset (*SIM*) in experiments. (1) *IDF* has 80 sensors recording an induced draft fan-machine's conditions, including total power and entrance pressure, with some correlated columns. (2) *NASDAQ* has 4 attributes: opening/closing, lowest/highest stock prices, with no clear relationships between them. Modifying predicates has limited impact on constraint performance. (3) *SIM*, derived from *IDF*, has 5 attributes with functional relationships. (4) *PUMP* consists of sensor data monitoring a pump, including 8 attributes. (5) *WADI* originates from an actual water distribution network, spanning 16 days in 2017, and records 60 attributes during both normal and attack scenarios.

Baselines & Implementation. We compare our bNDCRepair with several baselines, which are summarized in Table 3. The constraints we use for each dataset come from expert definitions or constraint mining algorithms. We insert errors into data based on [38, 41], focusing on continuous and correlated errors. Attribute boundaries are set using max and min values. Gaussian noise is added with default parameters $\delta = 2, \mu = -3, \omega = 10^{-3}$. According to Theorem 6.1, λ is set to $1.5 \cdot 10^{-7}$ for *IDF*, 10^{-7} for *NASDAQ*, 10^{-4} for *SIM*, 10^{-5} for *Pump* and 10^{-5} for *WADI*.

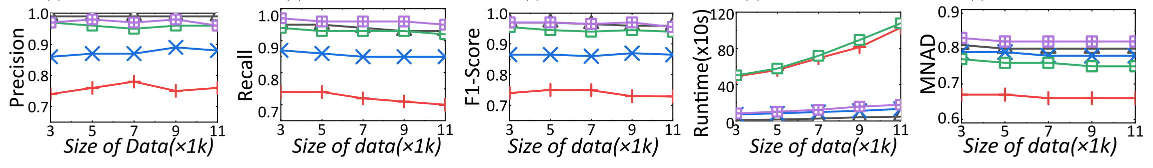
Metrics. The applied metrics include: (1) *Precision*(P) = $\frac{\#\text{correctIdentifyVio}}{\#\text{IdentifyVio}}$ measures the ratio between the number of violations correctly detected and the total number of violations detected by algorithms. (2) *Recall*(R) = $\frac{\#\text{correctIdentifyVio}}{\#\text{Vio}}$ is the ratio between #correctIdentifyVio and the total number of violations which actually happened. (3) *Mean Normalized Absolute Distance* (*MNAD*) $= 1 - \frac{\Delta(I_{\text{repair}}, I_{\text{truth}})}{\Delta(I_{\text{repair}}, I_{\text{noise}}) + \Delta(I_{\text{truth}}, I_{\text{noise}})}$ evaluates the data repair performance [25]. I_{noise} is the dataset to be repair, I_{truth} is the dataset without errors and I_{repair} denotes the data after repairing. The closer the *MNAD* value is to 1, the better the repair result.

Table 4: Overall performance comparison (Time (T for short) in Seconds).

	IDF					SIM					NASDAQ					Pump					WADI				
	P	*R*	*F1*	*T*	*MNAD*	*P*	*R*	*F1*	*T*	*MNAD*	*P*	*R*	*F1*	*T*	*MNAD*	*P*	*R*	*F1*	*T*	*MNAD*	*P*	*R*	*F1*	*T*	*MNAD*
oriRepair	0.69	0.62	0.65	**10.23**	0.41	0.65	0.49	0.56	**40.8**	0.33	0.67	0.53	0.59	**10.9**	0.58	0.6	0.62	0.61	**25.4**	0.47	0.58	0.53	0.72	**90.6**	0.29
Holoclean	0.72	0.67	0.69	310.3	0.43	0.67	0.74	0.70	845.2	0.38	0.59	0.66	0.62	1012.7	0.67	0.68	0.66	0.67	621.3	0.69	0.73	0.71	0.72	1720.1	0.53
Speed	0.71	0.64	0.67	164.8	0.62	0.75	0.69	0.72	524.7	0.57	0.62	0.73	0.67	750.5	0.71	0.71	0.68	0.69	250.9	0.57	0.56	0.52	0.54	601.4	0.4
CVtRepair	0.75	0.71	0.73	813.6	0.53	0.77	0.74	0.75	1982.7	0.68	0.69	0.7	0.69	1764.9	0.77	0.72	0.73	0.72	2199.4	0.7	0.78	0.74	0.76	2133.6	0.42
Clean4TSDB	0.89	0.85	0.87	112.1	0.78	0.82	0.9	0.86	433.2	0.81	0.78	0.71	0.74	664	0.89	0.85	0.82	0.83	344.7	0.75	0.77	0.78	0.77	978.2	0.49
bNDCRepair	**0.99**	0.93	0.96	41.4	0.8	**1**	0.84	0.91	228.8	0.82	0.87	0.91	0.89	72.3	0.87	0.88	0.91	0.89	156.8	0.77	0.93	**0.94**	0.93	307.7	**0.76**
bNDC+CVtRepair	0.96	0.93	0.94	893.3	0.75	0.83	0.8	0.81	2271.9	0.84	0.89	0.9	0.89	1824.8	0.82	0.81	0.91	0.86	2219.5	0.73	0.99	0.94	0.96	2451.9	0.48
bNDC+Clean4TSDB	0.98	**0.96**	**0.97**	150.9	**0.82**	0.91	**0.94**	**0.92**	659.1	**0.89**	**0.93**	**0.93**	**0.93**	577.2	**0.91**	**0.89**	**0.93**	**0.91**	443.7	**0.81**	**0.97**	0.92	**0.94**	365.3	**0.76**

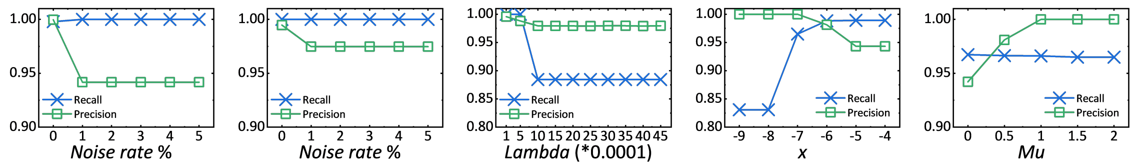


(a) Precision, *IDF* (b) Recall, *IDF* (c) F-1, *IDF* (d) Time, *IDF* (e) *MNAD, IDF*



(f) Precision, *IDF* (g) Recall, *IDF* (h) F-1, *IDF* (i) Time, *IDF* (j) *MNAD, IDF*

Figure 6: Comparison experimental result with varying error rate and data size



(a) I_{valid}, *IDF* (b) I_{valid}, *SIM* (c) λ, *SIM* (d) $\lambda = 10^x$, *NASDAQ* (e) varying μ, *NASDAQ*

Figure 7: Performance evaluation of bNDCRepair *w.r.t* varying parameters.

We report the overall performance (Sect 7.2), performance under two varying vital variables (Sect 7.3), discuss the impact of parameters in Sect 7.4, and propose discussion about the effectiveness of bNDC+CVtRepair in Sect 7.5.

7.2 Overall performance evaluation

Table 4 shows that the three algorithms that support both numerical value correction and precise constraint refinement, i.e., bNDCRepair, bNDC + CVtRepair, and bNDC + Clean4TSDB, achieve the best repair performance. In contrast, general-purpose data cleaning methods Holoclean and orirepair perform poorly, as they are neither designed for numerical data nor capable of modifying constraints. State-of-the-art sequential cleaning methods Speed and Clean4TSDB outperform the general approaches, yet their lack of support for constraint refinement limits their effectiveness compared to our proposed methods. Although CVtRepair allows for modification of imprecise constraints, it only supports predicate pruning and cannot precisely adjust thresholds, leading to suboptimal results. These findings show that constraint-based cleaning algorithms are highly sensitive to constraint accuracy, and thus precise constraint repair is essential for sequential data.

Our bNDCRepair ranks among the top in repair accuracy, maintaining an *F1-score* above 0.85 and achieving the highest repair precision, which confirms that refining imprecise constraints facilitates more accurate error identification. In terms of runtime, bNDCRepair is the most efficient among all methods except orirepair, achieving average speeds twice as fast as Clean4TSDB and an order of magnitude faster than CVtRepair.

From the dataset perspective, bNDCRepair performs consistently well even on complex datasets such as PUMP, which features high-frequency fluctuations, and WADI, which has large scale. This demonstrates the strong scalability and adaptability of bNDCRepair across different data environments. Furthermore, both bNDC + CVtRepair and bNDC + CVtRepair achieve comprehensive improvements in repair quality across all datasets compared to their original counterparts, despite sacrificing some runtime efficiency. This again highlights the importance of precise constraint refinement in enhancing data repair performance.

7.3 Effectiveness with varying error rate and data size

We evaluate the performance of several algorithms on the IDF dataset under varying anomaly rates and data scales, including CVtRepair (which supports predicate deletion), Clean4TSDB, our proposed bNDCRepair, and two hybrid approaches: bNDC + CVtRepair and bNDC + Clean4TSDB. The results are shown in Figure 6.

Our bNDCRepair consistently performs well across different noise rates and data scales, maintaining an *F1-score* above 0.95

Table 5: Contribution ratio with varying error rate.

	Error Rate	5%	10%	15%	20%	25%
IDF	bNDCRepair	0.66	0.71	0.72	0.94	1.29
	CVtRepair	0.34	0.29	0.28	0.06	-0.29
SIM	bNDCRepair	0.76	0.70	0.68	0.65	0.62
	CVtRepair	0.24	0.30	0.32	0.35	0.38
NASDAQ	bNDCRepair	0.80	0.75	0.78	0.81	0.82
	CVtRepair	0.20	0.25	0.22	0.19	0.18

and an *MNAD* value above 0.75, with minimal sensitivity to noise rates and dataset size. In terms of efficiency, bNDCRepair is the fastest, demonstrating stable performance across varying noise rates. Its runtime increases linearly with data size, and the growth rate is lower than that of the other algorithms, highlighting the strong scalability of our method. Moreover, the figures show that after incorporating the constraint refinement strategy proposed by bNDCRepair, the repair performance of both CVtRepair and Clean4TSDB improves across all noise rates and dataset sizes. In particular, their *F1* increase by more than 0.1, further demonstrating the critical role of precise constraint refinement in enhancing data repair quality for numerical sequential data.

7.4 Impact of critical parameters

We report the repairing result of bNDCRepair *w.r.t* three critical parameters, *i.e.,* the quality of the given validation set I_{valid}, the coefficient λ in $cost((\Sigma', I'), (\Sigma, I))$, and μ which adjusts the cost of constraint compression for Equation (1).

Evaluation of the quality of I_{valid}. Fig. 7(a-b) reports the Precision and Recall of bNDCRepair with varying error rate of I_{valid}. Since I_{valid} is involved in our method to avoid over-fitting and further fine-tune more accurate domain of constraints, there is an obvious drop in Precision when error rate reaches 1% on both datasets, while Recall is quite stable against the increasing error rates in I_{valid}. Here, we stress to apply a clean I_{valid} to provide reliable information of the data for repairing algorithms, which guarantees high repair accuracy with only a small size of correct data provided by human experts. In general, our proposed bNDCRepair achieves high-level F1-score with error rate less than 5% in I_{valid}, which verifies again that bNDCRepair has the ability to avoid both over-fitting and under-fitting.

Evaluation of λ. Fig. 7(c-d) shows the result on *SIM* and *NASDAQ* with varying λ. We note that the changes of inaccurate constraints is slightly compared with the repair of the whole dataset, as a consequence, the value of $cost_c(\Sigma, \Sigma')$ and $cost_d(I, I'|\Sigma)$ are not in the same order of magnitude! A higher λ value decreases the proportion of $cost_c(\Sigma, \Sigma')$ for our optimization objective, which may result in the under-fitting of constraint repairing. It shows that both the proper λ value and the tendency of Precision & Recall vary from different datasets. The repair performance stays stable when $\lambda > 0.001$ in *SIM*, while $\lambda > 0.001$ is a proper setting for *NASDAQ*. We conclude that λ is actually a important bridge to adjust the weight of constraint repair cost and data repair cost, which is consistent with our consideration when designing the objective function in Theorem 6.1. High-quality results can be expected from a good consistent order of magnitude of $cost_c(\Sigma, \Sigma')$ and $cost_d(I, I'|\Sigma)$ with a well-tuned λ.

Evaluation of μ. μ adjusts constraint expansion and compression costs. A larger μ encourages domain expansion, risking under-fitting. Fig. 7(e) shows *NASDAQ* results: as $|\mu|$ increases, bNDCRepair compresses the constraint domain, improving Precision and avoiding under-fitting. For $\mu > 0.5$, the domain stabilizes, with no further cost-effective expansions, maintaining repair performance.

Table 6: Contribution ratio with varying data size.

	#*DataSize*	10K	25K	50K	75K	100K
IDF	bNDCRepair	0.67	0.64	0.61	0.62	0.53
	CVtRepair	0.33	0.36	0.39	0.38	0.47
	#*DataSize*	3K	5K	7K	9K	11K
SIM	bNDCRepair	0.80	0.76	0.76	0.77	0.77
	CVtRepair	0.20	0.24	0.24	0.23	0.23
	#*DataSize*	2K	4K	6K	8K	10K
NASDAQ	bNDCRepair	0.43	0.77	0.60	0.60	0.63
	CVtRepair	0.57	0.23	0.40	0.40	0.37

7.5 Further discussion about bNDC+CVtRepair

From Table 4 and Figure 6, it can be observed that combining bNDCRepair with CVtRepair significantly improves the performance of the CVtRepair algorithm. We further analyze the bNDC+CVtRepair approach to investigate how the threshold for precise constraint refinement impacts the algorithm's effectiveness.

The contribution ratio among bNDCRepair **and** CVtRepair. Table 5 and Table 6 report the contribution ratios among bNDCRepair and CVtRepair. The ratios of the two methods are calculated by $\frac{\#\text{bNDCRepair.F1}}{\#\text{bNDCRepair.F1}+\#\text{CVtRepair.F1}}$ and $\frac{\#\text{CVtRepair.F1}}{\#\text{bNDCRepair.F1}+\#\text{CVtRepair.F1}}$, respectively. Table 5 shows that as error rate increases, bNDCRepair's contribution rises on *IDF*, falls on *SIM*, and stays stable on *NASDAQ*, while CVtRepair's trend is opposite. On *IDF*, where functional relationships are weak, CVtRepair's contribution becomes negative at a 25% error rate, highlighting bNDCRepair's importance. Table 6 reveals that on *IDF* and *SIM*, both algorithms' contributions stabilize with more data, indicating sufficient information for constraint modification. bNDCRepair consistently outperforms CVtRepair, confirming the need to update inaccurate constraint value domains.

8 CONCLUSION

This paper addresses repairing low-quality numerical data and inaccurate constraints. We introduce a cost function, propose two modification operations, and develop algorithms to balance over- and under-fitting. Theoretical analysis covers algorithm complexity and optimum solutions. Experiments show our method improves F1-score by 9.7%-17.6% and MNAD performance over methods without constraint updates. It also overcomes limitations of existing constraint repair methods. Combining our modifications with existing ones promises higher performance. Future work includes deeper theoretical research on data cleaning with multi-type constraints and developing continuous cleaning procedures with effective modification selection algorithms.

9 ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (2023YFB3308003), the National Natural Science Foundation of China (NSFC) (62572151, 62232005, 92267203), the Natural Science Foundation Project of Heilongjiang Province of China (YQ2024F005), and the Heilongjiang Provincial Young Talents Support Program for Science and Technology Innovation (2025QNTJ001).

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDB J.* 24, 4 (2015), 557–581.

[2] 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*. 925–936.

[3] George Beskales, Ihab F. Ilyas, Lukasz Golab, and Artur Galiullin. 2013. On the relative trust between inconsistent data and inaccurate constraints. In *29th IEEE International Conference on Data Engineering, ICDE*, Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou (Eds.). IEEE Computer Society, 541–552.

[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] Fei Chiang and Renée J. Miller. 2011. A unified model for data and constraint repair. In *Proceedings of the 27th International Conference on Data Engineering, ICDE*, Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan (Eds.). IEEE Computer Society, 446–457.

[6] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *Proc. VLDB Endow.* 6, 13 (2013), 1498–1509.

[7] Tamraparni Dasu, Rong Duan, and Divesh Srivastava. 2016. Data Quality for Temporal Streams. *IEEE Data Eng. Bull.* 39, 2 (2016), 78–92.

[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, Zekai Qian, Hongzhi Wang, Siying Chen, Yafeng Tang, Hongbin Su, Huan Hu, and Chen Wang. 2025. UniClean: A Scalable Data Cleaning Solution for Mixed Errors based on Unified Cleaners and Optimized Cleaning Workflow. *Proc. VLDB Endow.* 18, 11 (2025), 4117–4130. https://www.vldb.org/pvldb/vol18/p4117-wang.pdf

[10] Xiaoou Ding, Zekai Qian, Hongzhi Wang, Zhe Sun, Siying Chen, Hongbin Su, and Huan Hu. 2025. UniClean: A Multi-Signal Fusion Pipeline for Optimizing Data Cleaning Workflow. In *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 4600–4603. https://doi.org/10.1109/ICDE65448.2025.00362

[11] Xiaoou Ding, Yichen Song, Hongzhi Wang, Chen Wang, and Donghua Yang. 2024. MTSClean: Efficient Constraint-based Cleaning for Multi-Dimensional Time Series Data. *Proc. VLDB Endow.* 17, 13 (2024), 4840–4852. https://www.vldb.org/pvldb/vol17/p4840-wang.pdf

[12] 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

[13] 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

[14] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Zijue Li, Jianzhong Li, and Hong Gao. 2019. Cleanits: A Data Cleaning System for Industrial Time Series. *PVLDB* 12, 12 (2019), 1786–1789.

[15] 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

[16] Ju Fan and Guoliang Li. 2018. Human-in-the-loop Rule Learning for Data Integration. *IEEE Data Eng. Bull.* 41, 2 (2018), 104–115.

[17] Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management*. Morgan & Claypool Publishers.

[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] Lukasz Golab, Howard J. Karloff, Flip Korn, Avishek Saha, and Divesh Srivastava. 2009. Sequential Dependencies. *PVLDB* 2, 1 (2009), 574–585.

[20] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning*. ACM.

[21] Mingyue Ji, Xiukun Wei, and Dongjing Miao. 2020. Threshold Functional Dependencies for Time Series Data. In *DASFAA Workshops - BDMS*, Vol. 12115. 164–174.

[22] Rui Kang, Shaoxu Song, and Chaokun Wang. 2022. Conditional Regression Rules. In *38th IEEE International Conference on Data Engineering, ICDE*. IEEE, 2481–2493.

[23] 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.

[24] Kim-Hung Le and Paolo Papotti. 2020. User-driven Error Detection for Time Series with Events. In *36th IEEE International Conference on Data Engineering, ICDE*. 745–757.

[25] Qi Li, Yaliang Li, Jing Gao, Bo Zhao, Wei Fan, and Jiawei Han. 2014. Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In *SIGMOD*. 1187–1198.

[26] 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.

[27] 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.

[28] Mohammad Mahdavi and Ziawasch Abedjan. 2021. Semi-Supervised Data Cleaning with Raha and Baran. In *11th Conference on Innovative Data Systems Research, CIDR 2021,*.

[29] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *Proc. VLDB Endow.* 13, 3 (2019), 266–278.

[30] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.

[31] 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*. 1:1–1:7.

[32] 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.

[33] Shaoxu Song and Aoqian Zhang. 2020. IoT Data Quality. In *The 29th ACM International Conference on Information and Knowledge Management CIKM*. 3517–3518.

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

[35] Shaoxu Song, Han Zhu, and Jianmin Wang. 2016. Constraint-Variance Tolerant Data Repairing. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 877–892.

[36] Maksims Volkovs, Fei Chiang, Jaroslaw Szlichta, and Renée J. Miller. 2014. Continuous data cleaning. In *IEEE 30th International Conference on Data Engineering, ICDE*. 244–255.

[37] 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.

[38] Xi Wang and Chen Wang. 2020. Time Series Data Cleaning: A Survey. *IEEE Access* 8 (2020), 1866–1881.

[39] 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.

[40] Wei Yin, Tianbai Yue, Hongzhi Wang, Yanhao Huang, and Yaping Li. 2018. Time Series Cleaning Under Variance Constraints. In *Database Systems for Advanced Applications - DASFAA Workshops*. 108–113.

[41] Yang Zhang, Nirvana Meratnia, and Paul J. M. Havinga. 2010. Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *IEEE Commun. Surv. Tutorials* 12, 2 (2010), 159–170.