Akane: Perplexity-Guided Time Series Data Cleaning

XIAOYU HAN, School of Computer Science, Fudan University, China
HAORAN XIONG, School of Computer Science, Fudan University, China
ZHENYING HE*, School of Computer Science, Fudan University, China
PENG WANG, Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, China
CHEN WANG, National Engineering Research Center for Big Data Software, EIRI, Tsinghua University, China
X. SEAN WANG, School of Computer Science, Fudan University, China

Dirty data are prevalent in time series, such as energy consumption or stock data. Existing data cleaning algorithms present shortcomings in dirty data identification and unsatisfactory cleaning decisions. To handle these drawbacks, we leverage inherent recurrent patterns in time series, analogize them as fixed combinations in textual data, and incorporate the concept of perplexity. The cleaning problem is thus transformed to minimize the perplexity of the time series under a given cleaning cost, and we design a four-phase algorithmic framework to tackle this problem. To ensure the framework's feasibility, we also conduct a brief analysis of the impact of dirty data and devise an automatic budget selection strategy. Moreover, to make it more generic, we additionally introduce advanced solutions, including an ameliorative probability calculation method grounded in the homomorphic pattern aggregation and a greedy-based heuristic algorithm for resource savings. Experiments on 12 real-world datasets demonstrate the superiority of our methods.

CCS Concepts: • **Information systems** → **Data cleaning**.

Additional Key Words and Phrases: time series, data cleaning, recurrent patterns, perplexity

ACM Reference Format:
Xiaoyu Han, Haoran Xiong, Zhenying He, Peng Wang, Chen Wang, and X. Sean Wang. 2024. Akane: Perplexity-Guided Time Series Data Cleaning. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 121 (June 2024), 26 pages. https://doi.org/10.1145/3654993

1 INTRODUCTION

Time series data are rarely free of dirty values, such as sensor data from unreliable devices [27], energy consumption data measured by malfunctioning metering systems [29], or even stock and flight data [31] , which we believe to be fairly clean. For instance, concerning stock prices, some

*Corresponding author

Authors' addresses: Xiaoyu Han, xyhan22@m.fudan.edu.cn, School of Computer Science, Fudan University, Shanghai, China; Haoran Xiong, hrxiong20@fudan.edu.cn, School of Computer Science, Fudan University, Shanghai, China; Zhenying He, zhenying@fudan.edu.cn, School of Computer Science, Fudan University, Shanghai, China; Peng Wang, pengwang5@fudan.edu.cn, Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, Shanghai, China; Chen Wang, wang_chen@tsinghua.edu.cn, National Engineering Research Center for Big Data Software, EIRI, Tsinghua University, Beijing, China; X. Sean Wang, xywangcs@fudan.edu.cn, School of Computer Science, Fudan University, Shanghai, China.

ACM 2836-6573/2024/6-ART121
https://doi.org/10.1145/3654993

sources inaccurately associate the symbol SY with SALVEPAR rather than SYBASE because they share the same SY symbol in specific sources.

In the data analysis realm, such as classification [30], clustering [39] , and forecasting [13], the utilization of dirty data will compromise the analytical quality and render it unreliable. A straightforward solution involves employing anomaly detection algorithms to identify and discard anomalies. Regrettably, while handling a sole univariate time series, this approach will also undermine the analytical quality, particularly when considering consecutive dirty points [51]. Hence, time series data cleaning algorithms have been proposed and adopted to rectify dirty time points and improve data quality. Recent studies [40, 50] have proved their effectiveness for time series classification and forecasting tasks on various datasets.

Traditional time series data cleaning algorithms focus on smoothing technology, including Moving Average (MA) and AutoRegression (AR). MA determines a specific data point's cleaning result by calculating its adjacent points' weighted average. Despite simplicity, it tends to over-clean, as it modifies almost all data points, many of which were initially correct. AR fits a model with the given time series and applies it in reverse to generate predictions. A data point is considered dirty if it deviates significantly from the prediction, and the prediction is used for cleaning. Nevertheless, the dirty data will adversely impact the fitting process, undermining its efficacy.

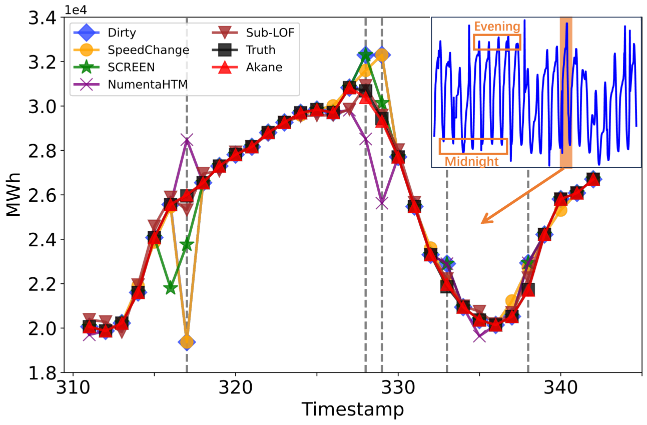

Fig. 1. Exemplary part of energy consumption data

In recent years, numerous powerful algorithms have been developed following the minimum change principle [5]. The constraint-based algorithm SCREEN [41] identifies and corrects points exceeding a single-speed constraint, which was later extended to accommodate multi-speed constraints [16]. Another representative statistics-based algorithm, SpeedChange [50], incorporates the likelihood cleaning concept from relational data cleaning and maximizes the speed change likelihood to clean. It exhibits a relatively high sensitivity in detecting slight deviations.

Besides, some prevalent time series anomaly detection methods can also be extended to data cleaning tasks. For instance, NumentaHTM [2] and FFT [34] can directly use the forecasting or reconstruction series to clean, while GrammarViz [38] or Sub-LOF [6] can refer to the mined grammar rules or nearest clean neighbors used for anomaly detection to clean.

However, existing cleaning algorithms have certain flaws. Let us start with a motivating example.

Example 1. *Figure 1 is California's hourly energy consumption data, zoomed in from the whole series' shaded part. A large spike dirty point at time 317, a consecutive error with two dirty points at times 328-329, and two small dirty points at times 333 and 338 are present marked by dashed lines.*

1) From the dirty data identification perspective, most methods make omissions or mistakes. SCREEN fails to detect small dirty points at times 333 and 338 since they do not violate the speed constraints, *while SpeedChange struggles to solve the large spike at time 317 due to its assumption of error ranges. Besides, for consecutive dirty points at times 328-329, they can at most identify and clean one of them due to the inherent limitations in their solutions. For Sub-LOF, it nearly regards all the shown data points as dirty because it uses sliding windows, not a single point, for anomaly detection. This makes a series of initially clean points adjacent to dirty points misidentified.*

2) From the cleaning decision perspective, most methods deviate a lot from the truth. SCREEN can merely change the dirty slump at time 317 just right to meet the speed constraint, which is still non-compliant with the actual upward trend. Another case occurs at time 338, where SpeedChange detects the dirt but adjusts its adjacent points to align with it. For NumentaHTM, the imprecise forecasting series makes the cleaning results at times 317 and 328-329 far from the truth, which can be attributed to the dirty data mixed into the data for model fitting.

3) Using perplexity, our method Akane can sensitively find the incongruent places with the context and identify them as dirty data, like the dirty points at times 317, 328-329, 333, and 338. Besides, the cleaning decisions also consider the proper contexts, and the statistics can separate clean and dirty data to avoid confusion, which can derive the decisions close to the truth combined with linear regressions.

Actually, existing time series data cleaning algorithms make only limited use of time series information during the cleaning process. Indeed, we notice that real-world time series data, whether exhibiting seasonality or not, often contain recurrent patterns. Some typical examples include weather data [23], daily CO2 intensity data [22], and GPS data [22]. This property finds application in numerous time series analysis tasks, such as anomaly detection [15], classification [17] , and forecasting [24]. Back to time series data cleaning, proper and satisfactory utilization of the recurrent patterns in time series can also be crucial. For instance, the energy consumption changes in the evening and midnight are evidently different recurrent patterns. It is usual to observe a significant rise in the evening, but such an increase occurring at midnight is nearly unheard of and definitely dirty. Unfortunately, existing algorithms struggle to distinguish between these two cases because the increase at midnight seems reasonable from a global perspective, leading to overlooking dirty points. By leveraging and analyzing the inherent recurrent patterns in the time series, we can clearly identify this contextually incongruent increase at midnight and make informed cleaning decisions using the regular midnight recurrent patterns.

However, implementing this idea poses significant challenges, primarily centered on using these recurrent patterns to determine which points belong to the dirty data and what their original values should be. Furthermore, it is also essential to strictly adhere to the minimum change principle throughout the whole process. In this paper, we propose a novel, statistics-based time series data cleaning algorithm that effectively uses the property of recurrent patterns.

Our intuition is that time series data exhibiting several recurrent patterns possess some similarities with textual data, as these recurrent patterns can be analogized to fixed word combinations. Subsequently, we introduce the concept of perplexity from Natural Language Processing (NLP) into time series analysis as the means to evaluate time series data quality. Comparable to textual errors, the presence of dirty data lowers the time series probability, thereby increasing its perplexity. Conversely, cleaning dirty data, which utilizes available statistics collected from the series itself, can be interpreted as an endeavor to reduce the time series perplexity within the given budget.

After formalizing the cleaning problem, we develop a four-phase framework to tackle the problem of perplexity computation and optimization for time series, given that the data points often exist within a continuous value space, in contrast to textual data. Besides the framework and selection strategies of concrete methods and parameters, we also mitigate concerns related to the potential adverse impact of dirty data on recurrent patterns and introduce an automatic yet highly effective budget selection strategy based on estimation, thus guaranteeing its feasibility. This rationale stems from the observation that the presence of dirty data can impede our ability to analyze recurrent patterns, and an improper cleaning budget can readily lead to either under or over-cleaning, thereby undermining our framework. Finally, we make advanced solutions to the framework to improve generalization capability. We introduce homomorphic pattern aggregation to generalize the probability calculation phase and present a greedy-based heuristic algorithm to save costs. Notably, patterns within time series data may sometimes exhibit similar behaviors while differing vastly in values. Inaccurate treatment of such patterns significantly impairs the method's applicability. Moreover, attaining the global optimum demands substantial time and space resources. These improvements make the framework more generic in both applicability and resource usage.

We summarize our major contributions in this paper as follows:

- We formalize the problem of cleaning time series data with recurrent patterns and introduce a comprehensive four-phase algorithmic framework, including concrete method and parameter selection strategies, for calculating and optimizing the perplexity within time series data.
- We mitigate the concerns about the adverse impact of the dirty data and devise an automatic budget selection strategy relying on estimating the gain derived from a typical cleaning case to ensure the framework's feasibility.
- We make advanced solutions to the framework, including a new probability calculation method based on homomorphic pattern aggregation to improve the applicability and a greedy-based heuristic algorithm for optimization to reduce resource utilization, making the framework more generic.
- We compare with 11 baselines on 12 real-world datasets, including 6 datasets with real dirty data, to prove the effectiveness of our works. In addition, we analyze each parameter's impact on our framework in detail. These also validate the method and parameter selection strategies.

2 PROBLEM STATEMENT

2.1 Preliminaries

2.1.1 Time Series. A *time series* is a sequence of data points ordered by time that can be denoted as the form $X = x[1], x[2], \cdots, x[n]$, where each $x[i]$ represents the value of the i-th data point with a corresponding timestamp $t[i]$, and $n = |X|$ is the *length* of the time series. For brevity, we use x_i and t_i to stand for $x[i]$ and $t[i]$.

A *subsequence* $x_i, x_{i+1}, \cdots, x_j$ is part of the whole time series X that can be abbreviated to $x_{i:j}$. We use μ^X and σ^X as the symbols of the *mean value* and *standard deviation* of X respectively. Thus in the *normalized time series* of X, denoted as $\hat{X}$, $\hat{x}_i = (x_i - \mu^X)/\sigma^X$.

Given a sorted list of numbers $Breakpoints = \beta_1, \beta_2, \cdots, \beta_{r-1}$, we can transform the value range from continuous numerical values to a finite set $W_s = \{w_1, w_2, ..., w_r\}$ with r discrete symbols. Suppose $\beta_0 = \min_{x_k \in X} x_k = X_{min}$, $\beta_r = \max_{x_k \in X} x_k = X_{max}$ to be two hypothetical breakpoints, then x_i can be symbolized as $x_i^w = w_a$ if $x_i \in [\beta_{a-1}, \beta_a]$, and X^w denotes the symbolized series.

2.1.2 Perplexity and Likelihood. In information theory, perplexity is a metric that gauges the predictive power of a probability distribution or model on a specific sample. It has been extended to multiple fields [9, 28]. Here, we borrow the definition of perplexity from the NLP field and redefine it in the context of time series. The *perplexity* of time series X with a chosen symbolized form X^w is

$$PP(X) = PP(X^w) = \sqrt[n]{1/P(X^w)}$$
$$P(X^w) = P(x_1^w x_2^w \cdots x_n^w) = P(x_1^w)P(x_2^w|x_1^w) \cdots P(x_n^w|x_{1:n-1}^w)$$

where $P(X^w)$ is the occurrence probability of X^w and n is the length of X. The probability calculation will be explained in Section 3.3.1.

Owing to the continuous value range of the original time series, direct calculation of its occurrence probability is not feasible. Therefore, we employ the corresponding symbolized time series instead. It is evident that different symbolization strategies applied to a specific time series will yield distinct perplexity values. Hence, before utilizing perplexity for calculation and optimization, it is essential to predetermine a consistent symbolization strategy.

Since the occurrence probability will become extremely small after cumulative multiplication, we use *likelihood* as a replacement for calculation. The likelihood $L(X)$ of time series X is

$$L(X) = L(X^w) = \log P(X^w) = \log P(x_1^w) + \log P(x_2^w|x_1^w) + \cdots + \log P(x_n^w|x_{1:n-1}^w)$$

By maximizing the time series' likelihood, we can achieve its minimum perplexity conveniently.

2.2 Problem Definition

Regarding time series data cleaning, our intuition suggests that it shares some similarities with the grammar correction task in NLP since recurrent patterns within time series data can be analogized to fixed word combinations in textual data. Dirty points can disrupt part of these patterns, leading to increased perplexity for the whole series. Hence, the cleaning problem can be defined as finding a cleaned time series X' with lower perplexity compared to the original time series X.

Furthermore, in data cleaning tasks, we often adhere to the minimum change principle [5], which favors a cleaned time series X' close to X. This principle can effectively prevent over-cleaning and yield the most probable result. Existing time series data cleaning works [41, 50, 51] follow the distance-minimal measure, in which the cleaning cost from X to X' is $\Delta_d(X, X') = \sum_{i=1}^{n} |x_i' - x_i|$. Nevertheless, recent research [42] shows that the cardinality-minimal measure is more precise and meaningful in this task. So our works consider the cleaning cost from X to X' as follows:

$$\Delta(X, X') = \|\{x_i \mid x_i \neq x_i',\ 1 \leq i \leq n\}\|$$

and for symbolized series, the cost is similar as it measures the total count of different symbols.

Since minimizing perplexity equals maximizing likelihood, we describe our cleaning problem as

Problem 1. *Given a time series X of length n, a sorted list of numbers $Breakpoints = \beta_1, \beta_2, \cdots, \beta_{r-1}$ for symbolization, and a cleaning budget δ, the cleaning problem is to find a cleaned time series X' that satisfies $\Delta(X, X') \leq \delta$ and the likelihood $L(X')$ under the given symbolization strategy is maximized.*

Clearly, a proper budget δ is vital to avoid under or over-cleaning. So, we must establish the right δ, as discussed later in Section 3.4.2.

3 ALGORITHMIC FRAMEWORK

To address the cleaning problem, we devise an effective four-phase algorithmic framework. We first overview it and then present a step-by-step explanation of each in this section.

3.1 Framework Overview

Figure 2 delineates the four phases of the framework: time series symbolization, probability calculation, likelihood optimization, and time series reconstruction. The time series symbolization converts each continuous numerical data point into a symbol of a finite set, facilitating the proper capture of recurrent patterns and enabling subsequent probability calculations. The probability calculation phase specifies the method for determining the probability of the symbolized time series based on statistics, essential for further calculations on likelihood. Likelihood optimization employs a recursive algorithm rooted in dynamic programming to maximize likelihood, through which a cleaned symbolized time series is obtained via retracement. The symbols within it are then transformed back to numerical values in the time series reconstruction phase, yielding the final cleaning result. This whole framework aims to solve Problem 1, and we use **Akane** to denote it.

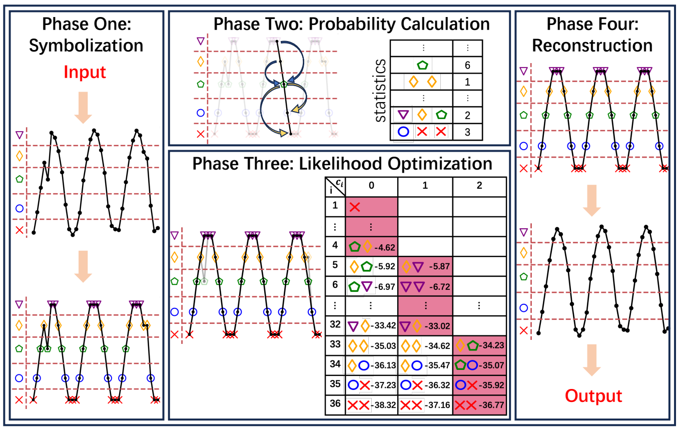
Phase One: Symbolization
Input
Phase Two: Probability Calculation
statistics
Phase Three: Likelihood Optimization
Phase Four: Reconstruction
Output

Fig. 2. Algorithmic framework overview

3.2 Time Series Symbolization

A suitable symbolization strategy must strike a balance. Inadequate symbolization sacrifices information and misses patterns, while excessive symbolization raises computational complexity and captures insignificant patterns, diminishing algorithm effectiveness.

One commonly used strategy is the SAX-Based [32] symbolization strategy. However, it primarily concentrates on the sequence matching task and is not well-suited for ours since it assumes that the normalized time series follows a Gaussian distribution. Thus, its breakpoints are designed to make the area under a $N(0, 1)$ Gaussian curve between each pair of neighbor breakpoints equal. However, time series data cleaning usually does not satisfy this premise.

We notice that symbolization is similar to the quantization process in analog signal digitization. In our task, a simple strategy is a uniform symbolization, which is versatile for almost any situation. To transform the value range from continuous numerical values to a finite set with cardinality r in time series X, we calculate the distance d between each two adjacent breakpoints by $d = (X_{max} - X_{min})/r$ and let the breakpoint $\beta_a = X_{min} + a * d$. Practically, data points in time series with recurrent patterns are usually non-uniformly distributed due to their akin real meanings. This is evident in Figure 1, where evening hours on weekdays consistently show a peak in energy consumption as people head home from work. Time series symbolization can indeed assign real meanings to data points. Hence, we draw inspiration from the optimal Lloyd-Max quantizer for a non-uniform quantization and use Lloyd's K-Means algorithm to symbolize. Through it, we can use prior distribution knowledge hidden in the time series, thus facilitating a more effective symbolization process.

The symbolization strategy to use can be decided based on the data property. For data exhibiting specific distribution characteristics, such as stable real-life data, K-Means better captures the inherent real-world meanings. Uniform symbolization is a foolproof choice when faced with uncertainty or unavailability regarding the data characteristics. To enhance cleaning effectiveness, it is often combined with pattern aggregation, as introduced in Section 4.1.

The number of clusters in K-Means, namely r, can be automatically determined by choosing the value corresponding to the minimum Davies-Bouldin Index (DBI), which is used to evaluate the clustering quality. In uniform symbolization, a too-small r will make it hard to capture pattern information, while a too-large one will result in the mistaken inclusion of some subtle yet insignificant changes as recurrent patterns. Here, we denote the mean of absolute speeds as $|\bar{v}|$ in the time series X, where $|\bar{v}| = \sum_{i=1}^{n-1} |v_i|/(n-1)$ and $v_i = x_{i+1} - x_i$. To capture the changes after symbolization, a reasonable choice is to set the distance d around the $|\bar{v}|$. In a loose scenario, we can let $d \approx 2|\bar{v}|$ to leave room for a subtle change, representing that usually, only two or more consecutive decreases or increases can be considered a change in real meaning. While in a strict one, we can let $|\bar{v}| \approx 2d$ to ensure an indisputable real meaning change hard to be aggregated. In this way, a possible r can be obtained from the range between $(X_{max} - X_{min})/(2 * |\bar{v}|)$ and $2 * (X_{max} - X_{min})/|\bar{v}|$. Experiments in Section 5.3 show that nearly any value within it can lead to a decent cleaning result.

3.3 Probability Calculation

3.3.1 Concrete Method. To calculate the likelihood, we must clarify how to calculate the probability.

Though straight, the simple assumption that the occurrence probability of a symbol is influenced by all its preceding symbols introduces considerable redundancy and results in probabilistic sparsity. Hence, to capture inherent short-term dependencies and local patterns and reduce calculation complexity, here we introduce the k-order Markov chain. That is, the occurrence probability of a symbol in a time series can be regarded as being determined solely by the last k symbols, then

$$P(x_i^w | x_{1:i-1}^w) = P(x_i^w | x_{i-k:i-1}^w)$$

The probability values can be computed using statistical information gathered from the original time series. Besides, we apply additive smoothing to tackle zero probabilities, that is

$$P(x_i^w | x_{i-k:i-1}^w) = \left(C\left(x_{i-k:i}^w\right) + 1\right) / \left(C\left(x_{i-k:i-1}^w\right) + r\right)$$

where $C\left(x_{i:j}^w\right)$ stands for the frequency of subsequence $x_{i:j}^w$ in symbolized time series X^w and r is the number of total available symbols. For simplicity, we can directly call $x_{i-k:i}^w$ a pattern here.

While a larger k may capture diverse patterns, it is unsuitable for our task because it makes dirty data more likely to exist in the $k + 1$ points used for calculation, which still results in probabilistic sparsity. Besides, it will also incur great time complexity. In fact, a direct choice of $k = 2$ or 3 is sufficiently ideal for us, and it has been adopted as a common solution for time series in previous work [10]. This has also been confirmed in our experiments in Section 5.3.2.

Example 2 (Time series symbolization and probability calculation). *For a dirty time series* $X = \{13, 6, 0, 12, 7, 1, 14, 7, 0, \underline{20}, 8, \underline{14}, 12, 6, 2, 11\}$ *with timestamps from* 1 *to* 16, *in which dirty points* 20 *and* 14 *occur at timestamps* 10 *and* 12, *whose ground truth are* 14 *and* 3, *we can first calculate the Davies-Bouldin index and obtain the most proper cluster number* 4. *Then we use K-Means to cluster it and get the centroids* $\{C_1 = 0.75, C_2 = 6.8, C_3 = 12.67, C_4 = 20\}$, *along with the breakpoints* $\{\beta_1 = 3.78, \beta_2 = 9.73, \beta_3 = 16.33\}$. *Through the breakpoints, we symbolize the time series to* $X^w = \{c, b, a, c, b, a, c, b, a, \underline{d}, b, \underline{c}, c, b, a, c\}$ *(here we use the alphabet). After fixing such a symbolization strategy, we can take* 2 *as the Markov chain order and calculate the probability of X as follows:*

$$\begin{aligned} P(X) &= P(c) * P(b|c) * P(a|cb) * \cdots * P(a|cb) * P(c|ba) \\ &= \frac{6+1}{16+4} * \frac{4+1}{6+4} * \frac{4+1}{4+4} * \cdots * \frac{4+1}{4+4} * \frac{3+1}{4+4} \end{aligned}$$

And we can get the likelihood $L(X)$ *by*

$$L(X) = \log\frac{7}{20} + \log\frac{5}{10} + \log\frac{5}{8} + \cdots + \log\frac{5}{8} + \log\frac{4}{8} \approx -12.45$$

We consistently use the unchanged statistics obtained from the original symbolized time series to ensure fairness in calculations.

3.3.2 Impact of Dirty Data. Given that we introduce the Markov chain into probability calculation, which makes the patterns analyzed much shorter, we must further guarantee that the recurrent patterns still exist in large quantities and appear more frequently than dirty patterns in the dirty time series. Otherwise, calculating probability in this way does not make sense.

Assuming that an infallible recurrent pattern of length $k+1$ has occurred m times, and the probability of a symbol being erroneously changed to one of the other $r-1$ symbols is cumulative $\tau = \delta/n$, which means each of them shares $\tau/(r-1)$. We use $k+1$ as the length because k is the Markov chain order, and these k points, along with the subsequent one point, affect one of the probability calculations. Regardless of the transformations from other patterns to the given one, it is clear that this pattern will occur $m * (1-\tau)^{k+1}$ times if introducing dirty points, which is usually sufficient for a recurrent pattern used for cleaning when k and τ is not too large.

However, with a large m, dirty data could also turn into fictitious recurrent patterns. It is not bad, as the cleaning process toward the truth is carried out budget by budget when multiple dirty data points are within the pattern. We only need to ensure that the occurrence times of the dirty patterns do not closely match or exceed the corresponding clean patterns, or they will impede data cleaning by preventing the dirty data from being converted to clean data and tainting the clean one. When introducing a specific dirty point at a particular position into the pattern above, the occurrence times of the dirty pattern is $m * (1-\tau)^k * (\tau/(r-1))$. Thus, if this happens, it satisfies

$$m * (1-\tau)^k * (\tau/(r-1)) \geq m * (1-\tau)^{k+1} \Rightarrow \tau \geq 1 - 1/r$$

This is obviously impractical. In the simplest case where $r = 2$, the dirty rate τ must surpass 0.5, a condition unattainable in a typical cleaning scenario [11, 12, 35, 43] because discarding the data is a more judicious choice. This conclusion also holds even in an extreme scenario where the probability of changing to a specific symbol is τ. Hence, the adverse impact of dirty data is limited.

3.4 Likelihood Optimization

3.4.1 Optimization Solution. Problem 1 is clearly an optimization problem reminiscent of the knapsack problem. Hence, we employ a pseudo-polynomial time algorithm based on Dynamic Programming (DP) to address it. It aims to find the symbolized time series with maximum likelihood.

Consider $x'^{w}_{1:i}$ to be a subsequence of the cleaned symbolized time series X'^{w} with the last k points $x'^{w}_{i-k+1}, \cdots x'^{w}_{i}$, and the cleaning cost to be $\Delta(x'^{w}_{1:i}, x^{w}_{1:i}) = c_i$, we can calculate the maximum likelihood $\Lambda\left(i, c_i, x'^{w}_{i-k+1:i}\right)$ of $x'^{w}_{1:i}$ with the recurrence equation

$$\Lambda\left(i, c_i, x'^{w}_{i-k+1:i}\right) = \max_{x'^{w}_{i-k} \in W_s} \left[\Lambda\left(i-1, c_{i-1}, x'^{w}_{i-k:i-1}\right) + \log P\left(x'^{w}_{i} | x'^{w}_{i-k:i-1}\right)\right]$$

where $c_{i-1} = c_i - 1$ if $x'^{w}_{i} \neq x^{w}_{i}$, otherwise $c_{i-1} = c_i$.

Initially, for $i = k$ and $\forall x'^{w}_{1} \cdots, x'^{w}_{k} \in W_s$, if $c_k \geq \Delta(x'^{w}_{1:k}, x^{w}_{1:k})$:

$$\Lambda\left(k, c_k, x'^{w}_{1:k}\right) = \log P(x'^{w}_{1}) + \cdots + \log P(x'^{w}_{k} | x'^{w}_{1:k-1})$$

else if $c_k < \Delta(x'^{w}_{1:k}, x^{w}_{1:k})$, we have it to be $-\infty$.

Through the above equations, we can obtain the maximum likelihood of the budget-allowed cleaned time series X' by calculating $\max_{x'^{w}_{n-k+1:n}} \Lambda\left(n, \delta, x'^{w}_{n-k+1:n}\right)$ where δ is the given budget. Algorithm 1 shows this optimization process. A recursive function `recurDP` is used in Lines 4 and 8 to implement the DP process with various loops, which receives the whole matrix Λ as input. In Line 5, `budgetSelect` is used to select the proper budget, as illustrated later in Section 3.4.2.

We can easily get the optimal cleaning result by retracing the intermediate calculations leading to the maximum likelihood, and the detailed approach is omitted here for simplicity.

Since only part of Λ is accessed during tracing, it incurs fewer time costs than Algorithm 1. Besides, the budget δ is considered to be proportional to n. Hence, the overall time complexity for getting the optimal cleaned result is $O(r^{k+1}kn^2)$, where $r = |W_s|$. Moreover, the space complexity amounts to $O(r^k n^2)$ primarily for storing Λ.

Algorithm 1: Maximum likelihood optimization

```
   Input: symbolized time series $X^w$ with length $n$ and Markov chain order $k$
   Output: maximum likelihood of the optimal result within the given budget $\delta$
1  Initialize $\Lambda(k, c_k, x'^w_{1:k})$ as mentioned above and let $\delta \leftarrow 0$.
2  for $c_i \leftarrow 0$ to $n$ do
3      for $i \leftarrow k+1$ to $n$ do
4          recurDP$(i, 0, c_i, k, k, \Lambda)$
5      if budgetSelect$(\Lambda, c_i, n, |W_s|, k)$ = True then
6          $\delta \leftarrow c_i$, Break
7  return $\max_{x'^w_{n-k+1:n}} \Lambda\left(n, \delta, x'^w_{n-k+1:n}\right)$
8  Function recurDP$(i, c_{i-1}, c_i, cur\_k, k, \Lambda)$:
9      if $cur\_k = k$ then
10         foreach $x'^w_i \in W_s$ do
11             recurDP$(i, c_i - \Delta(x'^w_i, x^w_i), c_i, cur\_k - 1, k, \Lambda)$
12     else if $cur\_k = 0$ then
13         foreach $x'^w_{i-k} \in W_s$ do
14             $l \leftarrow \Lambda\left(i-1, c_{i-1}, x'^w_{i-k:i-1}\right) + \log P\left(x'^w_i | x'^w_{i-k:i-1}\right)$
15             if $l > \Lambda\left(i, c_i, x'^w_{i-k+1:i}\right)$ then
16                 $\Lambda\left(i, c_i, x'^w_{i-k+1:i}\right) \leftarrow l$
17         return
18     else
19         foreach $x'^w_{i-(k-cur\_k)} \in W_s$ do
20             recurDP$(i, c_{i-1}, c_i, cur\_k - 1, k, \Lambda)$
```

3.4.2 Automatic Budget Selection. Algorithm 1 highlights the significance of the budget, as it plays a pivotal role in controlling the cleaning degree. To ensure the feasibility of the likelihood optimization phase, we propose how to decide δ automatically to avoid troublesome manual settings.

Our goal is to reduce perplexity, which equals elevating likelihood. The likelihood keeps growing (perhaps eventually stabilizing) with the ascension of budget δ. However, a higher likelihood doesn't guarantee lower RMS Error (our primary metric of cleaning introduced in Section 5.1.3; lower is better) as shown in Figure 3. As the budget increases from 0, the likelihood constantly rises, and the RMS Error continuously falls. At a budget of 140, the RMS Error is minimum, after which it begins to ascend steadily, but the likelihood still rises. This occurs due to over-cleaning.

In general, the likelihood growth rate, namely the likelihood increase per budget, slows down with a higher budget since we prefer to clean data points that would elevate more likelihood first. Figure 3 shows a small growth rate during over-cleaning, as most dirty points have already been identified and corrected. This allows us to transform selecting a proper budget into determining a growth rate threshold at which the likelihood increase should halt.

A straight way is to decide through observation or trial and error, but this can be troublesome in practice. So, we propose an automatic but effective budget selection strategy. Our insight is that we can estimate the likelihood increase of a common cleaning case and subsequently employ it as the threshold. Let us examine a simple case where a point is deemed dirty while other points around it remain clean. For instance, let x^w_{k+1} represent the dirty point with true value y^w_{k+1}, and the other points from x^w_1 to x^w_{2k+1} are clean. In this way, our cleaning aim is to change x^w_{k+1} to y^w_{k+1}.

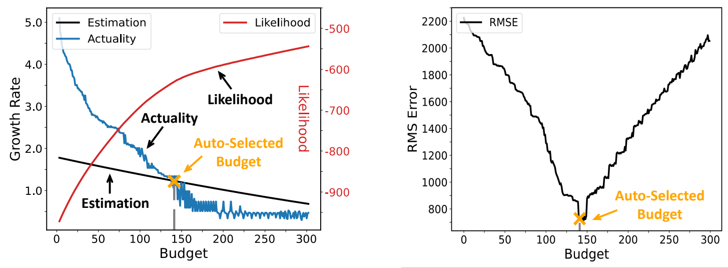

Fig. 3. Growth rate actuality, estimation, likelihood, and RMS Error of the cleaning result under budget δ

We suppose that the infallible subsequence appears m times, and the definitions of τ and k remain the same as Section 3.3.2. For simplicity, we estimate the likelihood increase from $P(x^w_{k+1}|x^w_{1:k})$ to $P(y^w_{k+1}|x^w_{1:k})$ at first. When introducing dirty points, $x^w_{1:k}$ will appear $m*(1-\tau)^k$ times. $x^w_{1:k+1}$ will appear $m*(1-\tau)^k*(\tau/(r-1))$ times. And $x^w_{1:k}, y^w_{k+1}$ will appear $m*(1-\tau)^{k+1}$ times. The estimated likelihood increase ω for cleaning the dirty point x^w_{k+1} will be

$$\omega = \log\frac{m(1-\tau)^{k+1}+1}{m(1-\tau)^k+r} - \log\frac{m(1-\tau)^k*(\tau/(r-1))+1}{m(1-\tau)^k+r} = \log\frac{m(1-\tau)^{k+1}+1}{m(1-\tau)^k*(\tau/(r-1))+1} \quad (1)$$

Actually, x^w_{k+1} will affect at most $k+1$ probability calculations. For the other calculations, the only dirty point x^w_{k+1} is on the right-hand side of the conditional probability. We can use the same way to estimate them, and the results are positive and close to ω when m is not great. To simplify our analysis and provide the tolerance for the case that dirty points are located at both ends and other complex cases, we can heuristically take ω as the threshold to solve this cleaning case. Obviously, if the threshold is smaller than ω, we might mistakenly modify many initially clean cases.

```
Algorithm 2: budgetSelect($\Lambda$, $\delta$, $n$, $r$, $k$)
Input: maximum likelihood matrix $\Lambda$, current budget $\delta$, time series length $n$, number of
       symbols $r$, and Markov chain order $k$
Output: a boolean value indicating whether to stop cleaning
1 if $\delta \geq k$ then
2     $\omega \leftarrow 0, \alpha \leftarrow 0$
3     for $c \leftarrow \delta - k + 1$ to $\delta$ do
4         $\tau \leftarrow \frac{c}{n}$
5         $\omega \leftarrow \omega + \log\frac{5*(1-\tau)^{k+1}+1}{5*(1-\tau)^k*(\tau/(r-1))+1}$
6         $\alpha \leftarrow \alpha + \max_{x'^w_{n-k+1:n}} \Lambda\left(n, c, x'^w_{n-k+1:n}\right) - \max_{x'^w_{n-k+1:n}} \Lambda\left(n, c-1, x'^w_{n-k+1:n}\right)$
7     if $\omega > \alpha$ then
8         return True
9 return False
```

When another error occurs in $x^w_{1:k}$, the frequency used in the denominator of Equation (1) will decrease to $m*(1-\tau)^{k-1}*(\tau/(r-1))$ and the numerators decrease similarly. We can estimate the new likelihood increase ω' for cleaning the dirty point x^w_{k+1} and ω' is obviously greater than ω. Adding the hypothesized dirty point in $x^w_{1:k}$ and conducting the estimation similar to Equation 1 on this hypothesized dirty point, the total likelihood increase exceeds $\omega' + \omega > 2\omega$ when two dirty points are in $x^w_{1:2k+1}$, which means the likelihood increase per point is still greater than ω. We can extend the conclusion to more dirty points and still take ω as the threshold.

Table 1. (a) Maximum likelihood L'_{Max} of cleaned series X' varying δ. (b) Detailed calculations of $\delta = 2$

δ	L'_{Max}
0	-12.45
1	-12.45
2	-10.95
3	-10.95
4	-10.95
$\cdots$	$\cdots$
$\cdots$	$\cdots$
$\cdots$	$\cdots$

(a)

i	<$x'^{w}_{i-1:i}$, L'_{Accuml}>	i	<$x'^{w}_{i-1:i}$, L'_{Accuml}>
1	<□c,-1.05>	9	<ba,-6.23>
2	<cb,-1.74>	10	<a$\underline{\text{c}}$,-6.93>
3	<ba,-2.21>	11	<$\underline{\text{c}}$b,-7.77>
4	<ac,-2.91>	12	<b$\underline{\text{a}}$,-8.24>
5	<cb,-3.75>	13	<$\underline{\text{a}}$c,-8.94>
6	<ba,-4.22>	14	<cb,-9.78>
7	<ac,-4.92>	15	<ba,-10.25>
8	<cb,-5.76>	16	<ac,-10.95>

(b)

Practically, to ensure the premise of the recurrent pattern when calculating the threshold and avoid considering some unnecessary pattern, we usually let $m = 5$ directly for convenience. Besides, a too-large m may make us overlook the potential cleaning case for a pattern from multiple dirty points to fewer dirty points. Moreover, since the dirty rate τ decided by budget δ is not known beforehand, our objective is to chase a balance between the budget and the growth rate. When the budget is low, the actual growth rate exceeds the theoretically calculated estimation a lot. We then incrementally increase the budget until the actual growth rate closely and continuously matches the theoretical lower bound. At this juncture, we halt the cleaning process and output the final result. This juncture is depicted in the mark in Figure 3, and the chosen budget is highly close to the optimum. To ensure a degree of robustness, the actual growth rate is determined using the interval mean. Algorithm 2 shows the selection process. The experiments in Section 5.3.4 fully validate it.

Example 3 (Detailed calculations of likelihood optimization, Example 2 continued). *After symbolizing and specifying the probability calculation method, we can calculate on the series* X^w.

Table 1 (a) shows the maximum $L(X')$*, namely* L'_{Max}*, as* δ *varies. Table 1 (b) exhibits the calculations leading to the maximum likelihood when* $\delta = 2$. L'_{Accuml} *is the accumulated optimal likelihood till the i-th points, and* $x'^{w}_{i-1:i}$ *is the* $i-1$*-th and i-th cleaned symbols. The likelihood remains constant when* $\delta = 1$ *because altering a single symbol cannot benefit. For instance, changing* x^w_{10} *from d to c can increase* L'_{Accuml} *till* $i = 10$ *by* $P(c|ba) - P(d|ba) = 0.69$*, but the total likelihood decreases by* 0.4.

As $\delta = 2$*, for* $i = k$*, the algorithm will propose a potential change of* x'^{w}_{k}*, and calculate the* L'_{Accuml} *till* $i = k$ *using the accumulated likelihood of each possible* $x'^{w}_{k-2:k-1}$ *till* $i = k - 1$ *and* $P(x'^{w}_{k}|x'^{w}_{k-2:k-1})$*. The details of the optimum are recorded in Table 1 (b). By retracing based on the recorded cleaned symbols, we can obtain the cleaning result of changing* x^w_{10} *to* $\underline{c}$ *and* x^w_{12} *to* $\underline{a}$*. The region* $\{\cdots, b, a, \underline{c}, b, \underline{a}, c, b, \cdots\}$ *containing these two points can elevate the likelihood by* 1.50*, resulting in a total likelihood increase from* -12.45 *to* -10.95*. At this moment, the* ω *in Algorithm 2 takes* $\tau = 1/16$ *and* $\tau = 2/16$ *into account and is* 2.87*, while the* α *is* 1.50 *getting from* $c = \delta = 2$*. Then, Algorithm 2 chooses* $\delta = 2$ *and stops the calculation, which is indeed the truth given that the likelihood remains unchanged from* $\delta = 2$.

3.5 Time Series Reconstruction

After the preceding three phases, we achieve a cleaned symbolized time series with maximum likelihood and minimum perplexity. Therefore, we finalize the cleaning process by reconstructing the time series from the symbols back to the numerical values.

Due to multiple regions from symbolization, a straight strategy is to select the centroid value within the corresponding region of a symbol. This is because points usually gather around the centroids (e.g., in K-Means), leading to an anticipated lower RMS Error.

However, this remains insufficient because it solely uses global information and neglects more worthwhile local one. Hence, we design a more powerful Linear Regression (LR)-based strategy.

Our idea involves using LR models to fit patterns found within the cleaned data points. Based on the surrounding initially clean data points and the fitted LR models, we can calculate the most likely values for reconstruction. For a cleaned data point, we first select a pattern it belongs to for fitting. To ensure adequate points for fitting and achieve better results, we choose the most frequent pattern containing at least half of the true data points at the cleaned data point's location. For all identical patterns in the whole time series, we set the values of data points at the same position as the cleaned data point in this pattern to be the model target and those corresponding to true data points as the model input. Of course, the number of samples should surpass the parameters for successful fitting. Then, we use these to fit the LR model and get the parameters. Finally, it is easy to calculate the reconstruction result of a cleaned data point through the true data points around it in the chosen pattern and the corresponding fitted model. If there are no suitable patterns or enough initially clean data points for fitting, this method can select the centroid value as a fallback.

Example 4 (Time series reconstruction, Example 3 continued). *After likelihood optimization and retracement, we get a cleaned symbolized series* $X'^{w} = \{c, b, a, c, b, a, c, b, a, \underline{c}, b, \underline{a}, c, b, a, c\}$*, whose likelihood increases from* -12.45 *to* -10.95*. If symbols remain unchanged, we keep them unchanged as well. For those altered, here we use the cleaned symbol* c *at timestamp* 10 *to illustrate the process.*

Since the Markov chain order is 2 *during our analysis, here we consider the patterns of length* 3 *containing timestamp* 10 *for reconstruction, which includes* $\{b, a, \underline{c}\}$, $\{a, \underline{c}, b\}$ *and* $\{\underline{c}, b, \underline{a}\}$*. We first exclude pattern* $\{c, b, a\}$ *even though it appears most frequently in the time series because there exist* 2 *dirty points at timestamp 10-12. Fitting a LR model with a single input cannot guarantee accuracy. For the rest of the patterns,* $\{b, a, c\}$ *appears* 3 *times while* $\{a, c, b\}$ 2 *times regardless of the dirty points. So we choose* $\{b, a, c\}$ *for fitting given that it appears most frequently, and* 3 *times are sufficient to fit the parameters we need. In the original time series, these* 3 *samples are* $\{6, 0, 12\}$,$\{7, 1, 14\}$ *and* $\{6, 2, 11\}$*. So our task becomes fitting the linear model* $y = \gamma_0 + \gamma_1 x_1 + \gamma_2 x_2$ *with model target* $\mathbf{y} = [12, 14, 11]$ *and the model input* $\mathbf{x} = [[6, 0], [7, 1], [6, 2]]$*. Through fitting we can get the parameters* $\gamma_0 = -3$, $\gamma_1 = 2.5$ *and* $\gamma_2 = -0.5$*, and for the pattern at timestamp* 10*, whose original value is* $\{7, 0, \underline{20}\}$*, we can get the cleaning result of* 20 *by calculating* $-3 + 2.5 * 7 - 0.5 * 0 = 14.5$*, which is close to the truth* 14 *compared to the centroid* 12.67*.*

We can use the same way to reconstruct the timestamp 12 *and obtain the cleaning result* 3.78*, and it is also better than* 0.75*. Finally,* $X' = \{13, 6, 0, 12, 7, 1, 14, 7, 0, \underline{14.5}, 8, \underline{3.78}, 12, 6, 2, 11\}$ *is the cleaned time series, which lowers the RMS Error from* 3.13 *to* 0.23*.*

Using the LR model makes sense because we usually have sufficient samples for fitting, and there exists scarcely any dirty data in these samples after cleaning, which enhances the fitting accuracy.

4 ADVANCED SOLUTIONS

The algorithmic framework elucidated in Section 3 can achieve satisfactory cleaning results on most occasions, primarily leveraging the recurrent patterns within univariate time series. Nevertheless, some limitations still exist with regard to applicability and resource consumption. Hence, in this section, we will make advanced solutions to the framework to make it more generic.

4.1 Homomorphic Pattern Aggregation

K-Means symbolization fails to fit specific cases, e.g., when a noticeable trend occurs as depicted in Figure 4 recording monthly air passengers time series. Though the ways to change within a year marked in Figure 4 behave similarly and can be the recurrent patterns, different base values due to the yearly growth in the passenger number make their values vary a lot. This presents an issue for our probability calculation phase since such patterns will be counted separately. We refer to these patterns as *homomorphic patterns*, and those behaving similarly belong to the same *homomorphic set*. To handle more cleaning situations, here we propose advanced strategies for our framework.

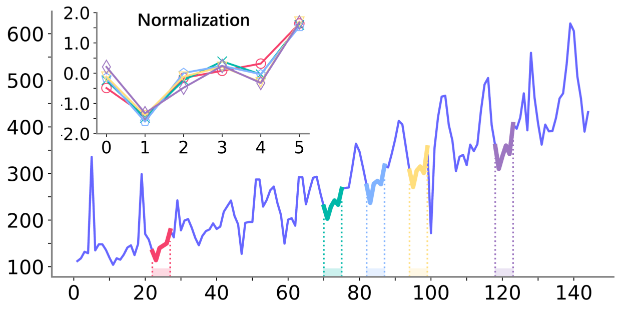

Fig. 4. Homomorphic patterns in the monthly air passengers time series

Uniform symbolization is a more practical approach when handling such uniformly distributed data. In uniform symbolization, more symbols are often necessary to capture most of the patterns, which makes a specific pattern typically manifest infrequently. We call this phenomenon *pattern sparsity*, and it will impede the employment of recurrent patterns for time series data cleaning. Hence, we introduce homomorphic pattern aggregation into probability calculation to diminish the negative influence of pattern sparsity and capture the hidden homomorphic patterns.

Due to the remarkable similarity in the behaviors of different patterns, here we consider time series normalization to eliminate the scale differences among them. It is suitable for solving certain trends, and its effect is shown in Figure 4. Given that what we consider are symbolized time series, we use the centroid value x_i^c of the symbolized data point x_i^w for calculation. Besides, we use the RMS Error between the normalized symbolized patterns to measure homomorphism. Suppose $x_{p-k:p}^w$ and $x_{q-k:q}^w$ to be two patterns with length $k+1$, then $x_{p-k:p}^w$ is homomorphic to $x_{q-k:q}^w$ if

$$Dist(x_{p-k:p}^w, x_{q-k:q}^w) = RMSE(\hat{x}_{p-k:p}^c, \hat{x}_{q-k:q}^c) \leq \epsilon$$

where ϵ is the distance threshold. Any two patterns in the same set are mutually homomorphic.

When performing probability calculation for a specific pattern, we aggregate all the patterns belonging to its corresponding homomorphic set for calculation. Hence, the core issue is how to discover the homomorphic sets for aggregated calculation using all the fixed-length patterns within the given time series. An ideal scenario is to minimize the number of sets, as this is more likely to result in better aggregation effects and lower complexity of subsequent calculation. For simplicity, we allow a pattern to exist in multiple homomorphic sets (but only a specific set is used for calculation). The homomorphic sets discovery problem can be formalized as:

Problem 2. *Given n patterns with a fixed length $m (m \ll n)$ and a threshold ϵ, the discovery problem involves identifying a list of homomorphic sets minimizing the total set number while covering all the given n patterns. Meanwhile, the distance between any two patterns within the same set is $\leq \epsilon$.*

Problem 2 is NP-Hard. We can prove it by reducing the NP-Hard minimum clique partition (MCP) problem in unit disk graphs (UDGs) [8, 14, 44] to it. Consider a UDG $G = (V, E)$ represented by a set S of n points in the plane, where the n vertices in V correspond to the n points, and an edge in E exists between two vertices if and only if the distance between the two points is at most 1. We can reduce the n points to n patterns, the plane to $m = 2$, the distance 1 to threshold $\epsilon = 1$, and the clique to the homomorphic set.

In this way, we propose a simple but effective way to predetermine all the homomorphic pattern sets that are adequate for our task. Algorithm 3 illustrates our idea of establishing the sets. We need to scan the patterns in the original symbolized time series sequentially. For a pattern, we either choose it as the base pattern to mark a homomorphic set, as shown in Lines 3-5 and Lines 10-12, or add it into the set of the nearest base pattern between which the RMS Error is less than $\frac{\epsilon}{2}$ as shown in Line 7-9, thus making the RMS Error between any two patterns within a set less than ϵ.

Algorithm 3: Heuristic construction of homomorphic sets

```
Input: symbolized series $X^w$, Markov chain order $k$, distance threshold $\epsilon$
Output: a dict $\mathcal{D}_{base}$ with base patterns as keys and homomorphic sets as values, a dict
        $\mathcal{D}_{set}$ recording existing patterns and corresponding base patterns and a k-d tree $\mathcal{T}$
        recording all base patterns
1  Initial $\mathcal{D}_{set}$, $\mathcal{D}_{base}$ and $\mathcal{T}$
2  for $i \leftarrow k+1$ to $n$ do
3      if $\mathcal{D}_{base}$ is empty then
4          $\mathcal{D}_{base}[x^w_{i-k:i}] \leftarrow x^w_{i-k:i}$, $\mathcal{D}_{set}[x^w_{i-k:i}] \leftarrow \{x^w_{i-k:i}\}$
5          Add $x^w_{i-k:i}$ in $\mathcal{T}$, Continue
6      Search 1-NN $c_{base}$ with RMS Error $c_{dist}$ of $x^w_{i-k:i}$ in $\mathcal{T}$
7      if $c_{dist} < \frac{\epsilon}{2}$ then
8          $\mathcal{D}_{base}[x^w_{i-k:i}] \leftarrow c_{base}$, Add $x^w_{i-k:i}$ in $\mathcal{D}_{set}[c_{base}]$
9      else
10         $\mathcal{D}_{base}[x^w_{i-k:i}] \leftarrow x^w_{i-k:i}$, $\mathcal{D}_{set}[x^w_{i-k:i}] \leftarrow \{x^w_{i-k:i}\}$
11         Add $x^w_{i-k:i}$ in $\mathcal{T}$
12 return $\mathcal{D}_{set}$, $\mathcal{D}_{base}$, $\mathcal{T}$
```

Sequentially searching the 1-NN base pattern requires the $O(kn)$ time. Here, we can use the k-d tree to accelerate this process. The average 1-NN searching time can be reduced to $O(k \log n)$, and the time complexity of the overall construction process is $O(kn \log n)$.

After obtaining the homomorphic sets, we can calculate the pattern probability by searching its 1-NN in the k-d tree, checking the dictionary to find its corresponding set, and then aggregating all the statistics. This incurs an average $O(k \log n)$ time cost since the size of the k-d tree is proportional to n, and the cardinality of each set is usually small, making the aggregation process neglectable.

For a pattern $x'^w_{p_i-k:p_i}$ in a set $\{x'^w_{p_1-k:p_1}, \cdots, x'^w_{p_s-k:p_s}\}$ of cardinality s, its occurrence probability is

$$P(x'^w_{p_i} | x'^w_{p_i-k:p_i-1}) = \frac{\sum_{j=1}^{s} C\left(x'^w_{p_j-k:p_j}\right) + 1}{\sum_{j=1}^{s} C\left(x'^w_{p_j-k:p_j-1}\right) + r}$$

When running the Algorithm 1, we only need to modify the probability calculation in Line 12, and the time complexity will increase to $O(r^{k+1}kn^2 \log n)$, with space complexity unchanged.

A proper threshold ϵ is crucial for pattern aggregation. Neither do we want to overlook essentially the same patterns nor aggregate dissimilar or slightly off dirty patterns. Fortunately, the normalization and mean square operation significantly mitigate the impact of variations across datasets, making it possible to use a fixed threshold ϵ. Experiments on 4 datasets, including 2 real-error ones, prove that a little threshold within 0.08, such as 0.02, can be a panacea.

4.2 Greedy-Based Heuristic Algorithm

Despite the effectiveness of our likelihood optimization phase, it incurs significant resource utilization in both time and space due to the global optimization. These unaffordable costs undermine our framework's applicability, not to mention the handling of homomorphic patterns. To make the whole framework more preferable, we refer to a simple approximation strategy for the knapsack problem, namely a greedy algorithm, and then design a heuristic algorithm to chase the likelihood increase in the likelihood optimization phase.

This algorithm is straightforward: since the global optimization problem aims to maximize the likelihood, we can greedily clean one point each time to achieve the maximum likelihood increase until reaching the automatically selected budget. Given that one point is cleaned each time, we need to focus on at most $k+1$ patterns of length $k+1$ that contain this point for likelihood increase.

Let X'^w be a current cleaned symbolized time series, initially $X'^w = X^w$, and X''^w be a symbolized cleaned time series after further cleaning a point $x_i'^w$ in X'^w to $x_i''^w$, while the other points are unchanged, then the i and $x_i''^w$ should be

$$\arg\max_{i, x_i''^w} \sum_{j=i}^{i+k} \left[\log P(x_j''^w | x_{j-k:j-1}''^w) - \log P(x_j'^w | x_{j-k:j-1}'^w) \right]$$

As to x_i'', its value is decided the same in Section 3.5.

The total number of cleaning candidates is $n * r$, and for each, at most $k+1$ patterns need to be calculated for likelihood increase. In fact, we can reuse some calculated likelihood increases because only one data point has changed each time, and the data points involved in most of the calculations remain the same. Specifically, after choosing a point to clean, at most $r*(k+1)$ candidates should be recalculated. So we can maintain a max-heap to store and reuse the likelihood increase information. It stores the modified symbol with the greatest likelihood increase of each data point. We need $O(rk^2n)$ time for first-round calculations and $O(n)$ for building max-heap. After cleaning a point, it takes $O(rk^3)$ time to calculate the changed scenarios and $O(k \log n)$ to update the max-heap. With a max-heap, the time complexity of the heuristic algorithm is $O(rk^3n + kn \log n)$, and the space complexity is $O(n)$ for max-heap. Considering homomorphic pattern aggregation, the time complexity will increase to $O(rk^3 n \log n)$, with the space complexity $O(kn)$. These are affordable since r and k are much smaller than n.

The greedy-based heuristic algorithm is prioritized when focusing on efficiency. It represents a trade-off between precision and time complexity. It can be gradually extended to consider the maximum likelihood increase from simultaneously cleaning two, three, or even more points until it degenerates to global optimization.

5 EXPERIMENTS

In this section, we select three methods from our algorithmic framework to evaluate its overall performance, namely **Akane** (K-Means symbolization + global optimization, no pattern aggregation), **AkaneH** (K-Means symbolization + heuristic algorithm, no pattern aggregation), and **AkaneH+** (uniform symbolization + heuristic algorithm, with pattern aggregation). We will introduce the detailed settings, compare them with 11 baselines on 12 real-world datasets to check whether our proposed methods are truly effective, and conduct thorough parametric analyses to find out how parameters affect our methods. Besides, we want to confirm whether the proposed strategies of method and parameter selection are valid. The relevant artifacts are provided in the GitHub [1].

5.1 Experimental Settings

5.1.1 Dataset. Table 2 shows the details of 12 real-world datasets used in our experiments. CA and Romania are datasets varied from the raw data regarding data length and dirty rate, rendering them well-suited for scalability evaluation. Since their raw data are both non-stationary, the CA dataset is concatenated by the extracted stationary parts to make us have both a stationary and a

[1]https://github.com/HatsuneHan/Akane-sigmod24

Table 2. Details of 12 real-world datasets

Name	Dirty Type	Length	Rate	Source	Notes
CA	Synth	1K-10K	5-30%	California ISO	Energy
Romania	Synth	1K-10K	5-30%	Kaggle	Energy
Product	Synth	397	20%	FRED	Utilities
Retail	Synth	377	20%	FRED	Trade
Passenger	Synth	144	20%	Kaggle	Airline
Traffic	Real	200	2%	Dodgers [3, 26]	Flow
ID_7c18	Real	1500	0.73%	IOPS [36]	KPI
ID_7698	Real	1K	0.2%	IOPS [36]	KPI
ID_a40b	Real	10K	0.32%	IOPS [36]	KPI
MSL-T-5	Real	2218	0.23%	NASA-MSL [25]	Telemetry
SMAP-G-2	Real	7361	0.014%	NASA-SMAP [25]	Telemetry
Stock	Synth	12824	10%	SCREEN [41]	Finance

non-stationary energy dataset for better evaluation, avoiding limitations resulting from a single scenario. Moreover, the Product, Retail, Passenger, and Stock datasets are fixed in terms of length and dirty rate. We take the datasets above as ground truths and inject synthetic dirty data into them. We consider a single big dirty point (20%), a single small dirty point (60%), and consecutive dirty points (20%), following the types of dirty data as well as injection methods summarized in related works [41, 45, 50, 51], and adhering to a reasonable distribution.

The remaining 6 datasets are also fixed and have real dirty data, as they come from the prevalent time series anomaly detection benchmarks [3, 25, 26, 36, 37], and their original forms consist of data and binary *is_anomaly* labels. We regard the anomalies as dirty and invite the domain experts to manually clean them, thus making them applicable to our cleaning task. It is noteworthy that some sources, such as IOPS [36] and Dodgers [3, 26], are so large that we use the sliced fragments.

5.1.2 Baselines. We incorporate a total of 11 methods as our baselines. Five of them are established data-cleaning algorithms, including EWMA [18], AR-Linear, AR-LSTM (AR algorithm using different models), SCREEN [41] and SpeedChange [50]. Since explicit speed constraints may not exist, we give SCREEN the ground truth to obtain them. The remaining six are extended from prevalent unsupervised anomaly detection algorithms involving forecasting-based Torsk [20], NumentaHTM [2], TripleES [1], reconstruction-based FFT [34], encoding-based GrammarViz [38] and distance-based Sub-LOF [6]. For Torsk, NumentaHTM, TripleES, and FFT, we directly use the forecasting/reconstruction results to clean. GrammarViz or Sub-LOF use the mined grammar rules or relative density to neighbors for anomaly score calculation. Hence, we use the best match rule or the nearest clean neighbor as the cleaning decisions. Since a point may be covered by multiple rules or neighbors in different subsequences, we choose the one closest to the ground truth.

5.1.3 Criteria. Root Mean Square Error (RMS Error) is widely employed to evaluate the quality of the time series data cleaning algorithms in many related works [41, 50, 51]. It measures the differences between truth and cleaned time series. Here, we choose it as the criteria in our experiments. Let X^{truth} be the ground truth and $X^{cleaned}$ be the cleaned time series, then it can be calculated by

$$RMSE(X^{truth}, X^{cleaned}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i^{truth} - x_i^{cleaned})^2}$$

where n is the series length, and a low RMSE is desired. Here we use $RMSE^{cleaned}$ for abbreviation.

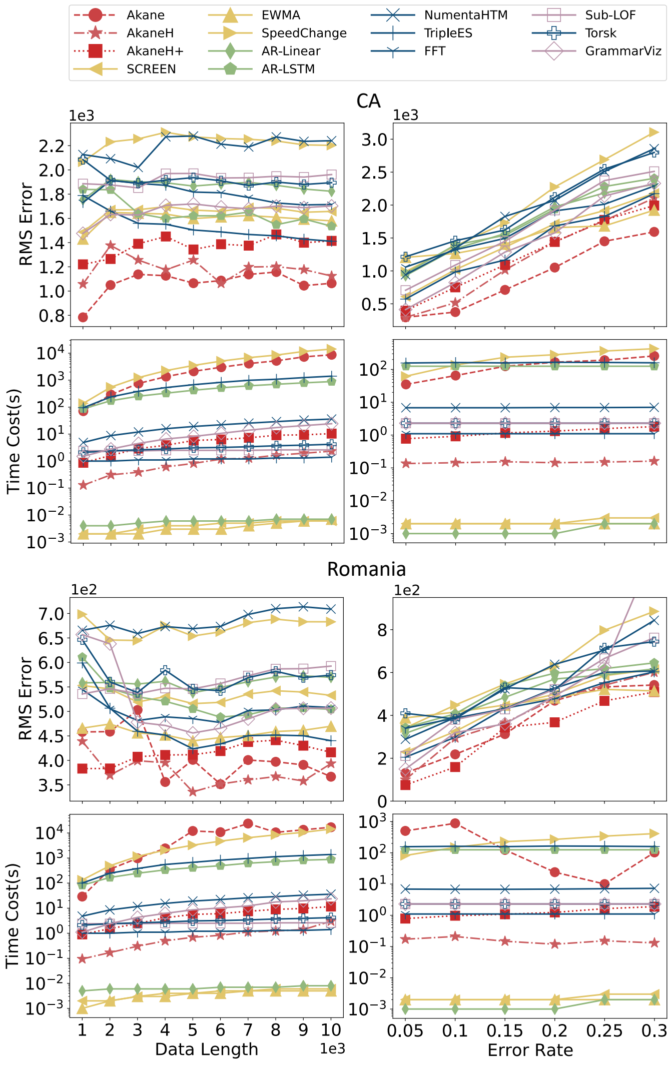

Fig. 5. Performances and time costs on variable CA and Romania datasets

5.2 Comparisons with Existing Methods

While conducting comparisons, our methods Akane, AkaneH, and AkaneH+ employ parameter selection strategies as illustrated in Section 3, that is, minimum DBI for K-Means, a value in the calculated range for uniform symbolization (for simplicity, $r = 25$ for all datasets), budget selected by Algorithm 2, and a small distance threshold within 0.08 (for simplicity, $\epsilon = 0.02$ here for all datasets). For baselines, we adhere to the strategies outlined in the origins, take the real meanings of different parameters into account, and endeavor to optimize their performance to set the parameters.

5.2.1 Comparisons on Variable Datasets. Figure 5 summarizes the performances of cleaning methods and their time costs on the CA and Romania datasets while varying length and dirty rates. EWMA performs well, especially at high dirty rates, as it can notably mitigate the adverse impact of the dirty data points. However, it tends to alter many original clean data points, while our methods strictly follow the minimum change principle and still excel in performance.

The LSTM model, capable of processing more intricate information, usually surpasses the Linear model in the AR algorithm. Nevertheless, AR algorithms demand more clean data for fitting to obtain enhanced cleaning results. As the dirty rate increases, the fitted model becomes less precise, but the impact of incorrect identifications might be diluted, rendering its performance erratic.

Similar things happen with anomaly-detection-based methods. Regarding dirty data identification, Sub-LOF, GrammarViz, and Torsk tend to assign identical and high anomaly scores for a window containing dirty points, not a single dirty point, to distinguish anomalies (or dirty data). While this is common in anomaly detection, it also stands for altering many initially clean data points in data cleaning tasks. Besides, the dirty data will also reduce the detection models' precision, especially in NumentaHTM, TripleES, and FFT, worsening the cleaning decisions. These result in their inferiority, not to mention violating the minimum change principle. FFT, TripleES, and GrammarViz exhibit relatively strong performance among them. The efficacy of FFT is attributed to its cleaning process, which uses the reconstruction result that deliberately discards insignificant information. In addition, TripleES benefits from the concept of smoothing during the identification process, assigning a notably high score to the exact dirty data for precise identification, and it can broadly weaken the impact of dirty data like what EWMA does. Moreover, GrammarViz can summarize relatively effective rules from the data to clean, but its performance sometimes slumps when the data is short, or the dirty rate is high despite the usage of ground truth. Our methods do not confound dirty data points with the truth, and the symbolization phase enables us to encapsulate primary information without becoming entrenched in specific values, thus having superior performance.

SCREEN's performance is relatively stable, albeit not as optimal as ours. It is crucial to note that SCREEN employs the ground truth, implying that the reported performance is theoretically optimal. Regrettably, the speed constraint limits its ability to identify dirty data within the constraint and clean them close to the truth, making it inferior to our methods.

Though based on statistics, SpeedChange performs poorly due to inherent method limitations and required parameters. It assumes an error range for dirty data, but determining this parameter is challenging when dealing with various types of dirty data, leading to subpar results. Besides, speed change information alone is insufficient to obtain cleaning results close to the truth. Our statistics-based methods do not make such a strong assumption about the dirty points, and they leverage more powerful information about recurrent patterns, facilitating superior cleaning results.

In Figure 5, Akane, AkaneH, and AkaneH+ exhibit notable performances. Akane and AkaneH usually perform better because the specific distribution characteristics present in real-world energy consumption data render K-Means symbolization a more proper choice. On the CA dataset, they demonstrate stable performance owing to sufficient recurrent patterns. Akane outperforms the others due to global optimization, with AkaneH closely following it. Though AkaneH+ occasionally misidentifies slightly off dirty patterns as clean and incorporates redundant information due to homomorphic pattern aggregation, which somewhat hampers its performance, it still surpasses the baselines. Remarkably, at a high dirty rate, AkaneH+'s performance approaches that of AkaneH due to the more accurate budget selection, as shown in Section 5.3.4.

On the Romania dataset, AkaneH+ performs the best on a short series having a singular trend. As the series lengthens and pattern occurrences increase, both Akane and AkaneH improve. The fluctuating trend within the non-stationary Romania dataset causes performance variations in Akane and AkaneH, and reduces the performance gaps resulting from symbolization strategies between them with AkaneH+. This trend, along with global optimization, sometimes makes Akane over-clean, mistakenly changing certain consecutive clean data points of lower probability to higher probability patterns, leading to a slight performance decline. In some cases, it may even perform worse than AkaneH. However, all three methods consistently outperform the baselines.

Upon varying dirty rates, our methods excel at low rates, as most patterns remain clean, making it easier to identify and rectify dirty points. As the dirty rate rises, there is a slight degradation in the performance, which is attributed to the excessive dirty points that impede the pattern statistics and adversely affect the reconstruction phase. Given that the dirty rate will not be too high in practical scenarios [11, 12, 35, 43] and our methods still retain commendable cleaning capabilities at that time, this is not of significant concern.

Regarding time costs, EWMA, SCREEN, and AR-Linear run faster due to their single sequential scans of the time series, hardly leveraging statistics from it. AR-LSTM is slower because of its iterative model refinement. Among ours, Akane outperforms SpeedChange with better cleaning results. We should notice that distinct auto-selected symbolization numbers of K-Means may make the time costs oscillate. By adopting heuristics, AkaneH and AkaneH+ greatly cut time costs. They balance performance and efficiency, as they usually run faster than most baselines yet still outperform all the listed methods except Akane. They are suited for scenarios prioritizing efficiency.

5.2.2 Comparisons on Fixed Datasets. The 10 fixed datasets are ideal for evaluating the overall cleaning abilities in different cases. Given the diverse scales across different datasets, we adopt *accuracy* as a metric for better presentation effects. It is defined as:

$$accuracy = \left(RMSE^{dirty} - RMSE^{cleaned}\right) \Big/ \left(RMSE^{dirty} - RMSE^{cleaned}_{min}\right)$$

where $RMSE^{cleaned}_{min}$ is the optimal RMS Error obtained by one of the methods. The best method yields an $accuracy = 1$, while $accuracy \leq 0$ if a method exhibits no or even adverse effects.

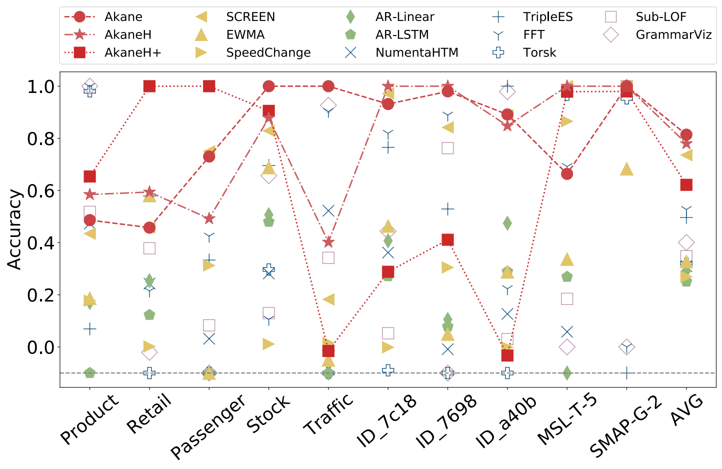

Fig. 6. Performance on 10 fixed datasets and the average

Figure 6 shows the performance of 14 different methods on 10 fixed datasets (algorithms that fail to run correctly are partially omitted). EWMA shows limited efficacy, while AR algorithms often conduct inaccurate fits and mistakenly modify many initially clean data points. The performance of 6 anomaly-detection-based methods is also mediocre, which shares similar reasons regarding undue identification and imprecise cleanings with Section 5.2.1. FFT, TripleES, and GrammarViz exhibit relatively strong performance among them, aligning with the outcomes and analyses on the aforementioned variable CA and Romania datasets in Section 5.2.1.

SCREEN performs well by leveraging ground truth to identify large spikes. SpeedChange can identify and rectify some dirty points but may confuse them with clean data due to its speed change assumption, leading to subpar performance.

Our methods generally perform better overall than existing methods (except SCREEN, which uses the ground truth and never produces an accuracy below 0). It is worth noting that, for the Product dataset, there is a consecutive error of length 8, which, along with the uniform distribution of the data, makes our methods' accuracies relatively low. For the Traffic, ID_7c18, ID_7698, and ID_a40b datasets, AkaneH+ performs poorly because many small and insignificant changes in the data make it difficult to capture meaningful recurrent patterns and select precise budgets. For the Product, Retail, and Passenger datasets, AkaneH+ outperforms Akane and AkaneH since singular trends exist in these datasets, proving the effectiveness of employing homomorphic pattern aggregation to detect and address covert deviations.

Combined with the former analyses, Akane and AkaneH, using K-Means symbolization, are better suited for real-world datasets with specific non-uniform distribution characteristics. Akane prioritizes high efficacy, whereas AkaneH, exploiting the idea of greed, emphasizes efficiency while retaining a large portion of efficacy. AkaneH+, using foolproof uniform symbolization and homomorphic pattern aggregation, is fit for data without such distribution or much prior knowledge. It prefers efficiency like AkaneH and shows particular proficiency in data having a singular trend.

5.3 Parametric Analysis

In parametric analysis, we assess the impact of the symbolization number r, Markov chain order k, and the homomorphic pattern distance threshold ϵ on the performance. Here, we follow the common settings of the length and dirty rate in related works, choosing 2 time series of length 1500 from both the CA and Romania datasets for analysis, with a dirty rate of 0.2, along with 2 datasets ID_7c18 and ID_7698 with real dirty data. We also evaluate our budget selection strategy using variable CA, Romania, and all of the 10 fixed datasets. When analyzing a specific parameter, we keep other parameters the same as the settings in Section 5.2 and unchanged, and we use optimal cleaning results without the automatic budget selection strategy to prevent potential interference.

5.3.1 Varying Symbolization Number r. Figure 7 reports the algorithm performances using K-Means for symbolization and the corresponding DBI with varying r. Employing $r = 6$ for dirty CA and Romania time series, $r = 3$ for ID_7698, and $r = 2$ for ID_7c18, which yields the minimum DBI, consistently produces nearly optimal cleaning results for both Akane and AkaneH, validating our proposed approach for selecting the symbolization number r in K-Means. It is essential to clarify that DBI cannot directly predict performance. A high DBI, though suggesting a comparatively low clustering effect, does not necessarily imply poor performance, as seen in Figure 7 when r equals 5 in CA and Romania. This is because the quantity of r itself can also affect the pattern characterization ability. For instance, although DBIs remain stable when r ranges from 9 to 15, our cleaning algorithms' effectiveness declines due to the pattern sparsity caused by a larger r.

For AkaneH+ using uniform symbolization, selecting the optimal r is challenging. Slight deviations in symbolization under two adjacent numbers can significantly impact the pattern characterization, leading to varied performances, as depicted in Figure 8. However, AkaneH+ performs decently within the range specified in Section 3.2 (indicated by the shadows in Figure 8, and the dashed lines represent $d = |\bar{v}|$). This proves that the proposed range aligns with a moderate selection of r, striking a balance that avoids pattern capture challenges while excluding insignificant changes, which shows the parameter tolerance of uniform symbolization.

5.3.2 Varying Markov Chain Order k. The Markov chain order k vastly affects performance, as depicted in Figure 9. Akane and AkaneH excel when $k = 2$ or 3, whereas AkaneH+ usually performs best with $k = 3$ (sometimes 4), which is entirely consistent with our selection strategy for k.

When k is 1, Akane and AkaneH show slightly lower performance, struggling to capture intricate recurrent patterns in the series. AkaneH+ also faces obstacles in homomorphic pattern identification

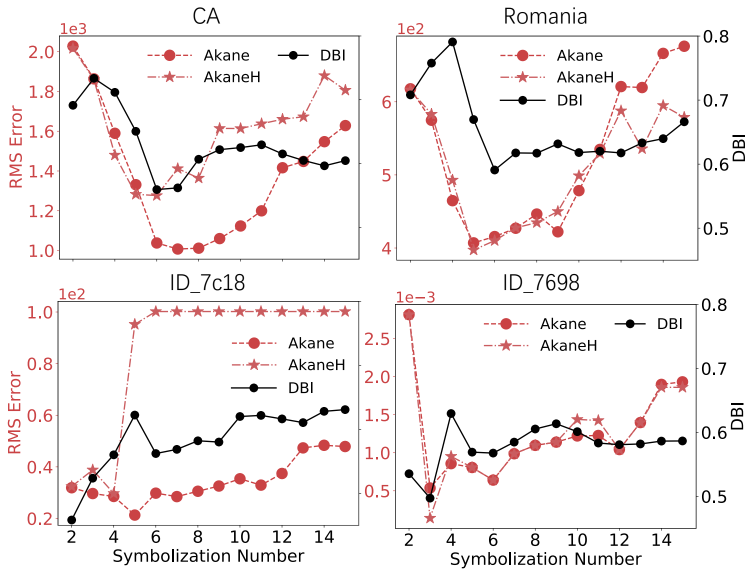

Fig. 7. Vary K-Means symbolization number r over 4 datasets with other parameters unchanged

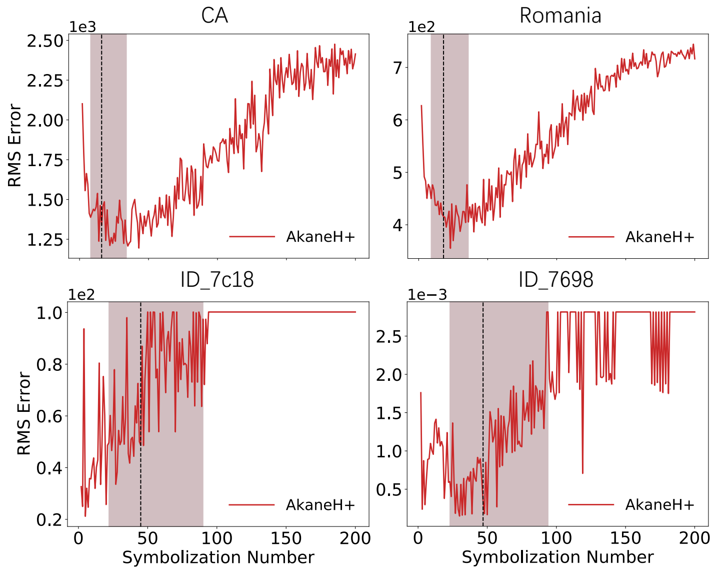

Fig. 8. Vary uniform symbolization number r over 4 datasets with other parameters unchanged

and aggregation. As k exceeds 4, it conforms to our analysis in Section 3.3.1, leading to poor cleaning ability due to probabilistic sparsity. Akane is impacted the least due to global optimization.

5.3.3 Varying Distance Threshold ϵ. Figure 10 reports how the performance of AkaneH+ changes with varying ϵ. When ϵ is set to 0, indicating no pattern aggregation, the algorithm usually exhibits poor cleaning results, confirming pattern sparsity under uniform symbolization. The result on ID_7698 is an exception owing to its low dirty rate and simple type of dirty data.

Experiments reveal that AkaneH+ performs well on all 4 datasets within the ϵ range of 0.01 to 0.08, confirming that an optimal ϵ is little influenced by datasets thanks to the normalization and

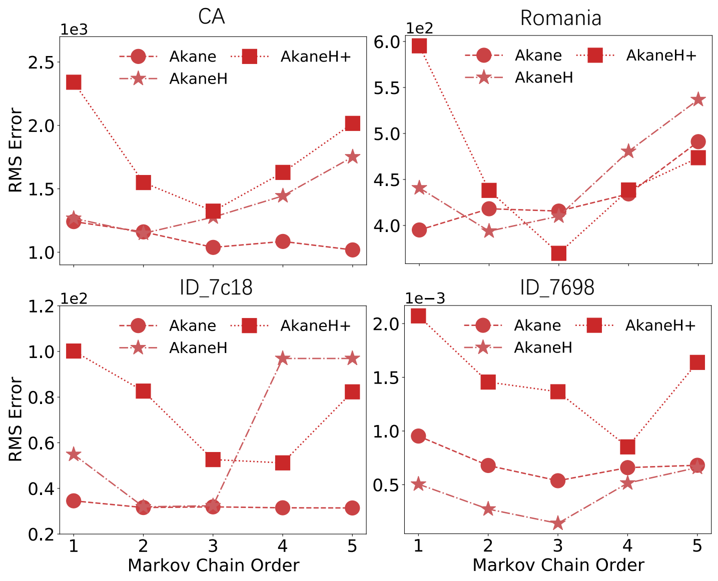

Fig. 9. Vary Markov chain order k over 4 datasets with other parameters unchanged

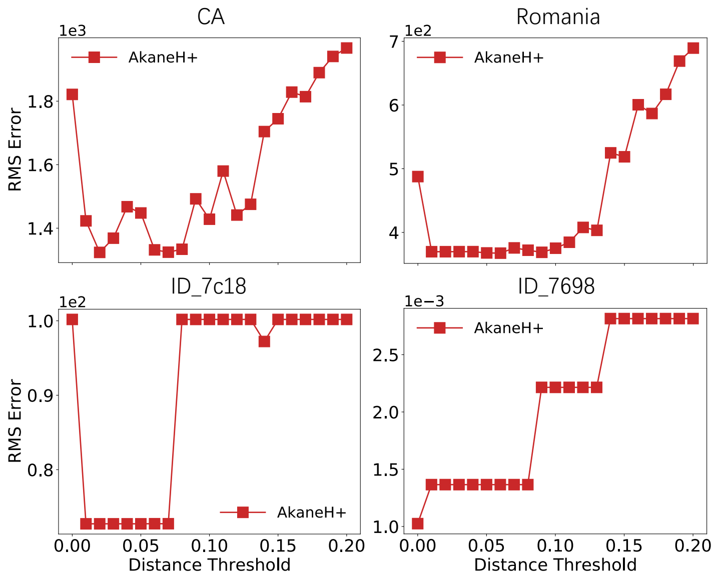

Fig. 10. Vary distance threshold ϵ for pattern aggregation over 4 datasets with other parameters unchanged

mean square operation. This range occurs because similar patterns have much smaller distances than entirely distinct patterns, ensuring stable performance across a broad range. It also enables AkaneH+ to avoid potential problems outlined in Section 4.1. It is noticeable that AkaneH+'s performance fluctuates within the CA dataset, possibly due to the aggregation of some unimportant changes before the actual scaled recurrent patterns. However, its adverse impact is limited.

5.3.4 Evaluation of Automatic Budget Selection Strategy. Table. 3 shows the accuracy of the automatic budget selection strategy on 2 variable and 10 fixed datasets. The accuracy is similar to that in

Table 3. Accuracy of automatic budget selection strategy

Evaluation	Akane	AkaneH	AkaneH+
Vary CA Length	0.953	0.944	0.966
Vary Romania Length	0.922	0.947	0.973
Vary CA Rate	0.974	0.894	0.939
Vary Romania Rate	0.892	0.853	0.991
10 fixed datasets	0.829	0.833	0.811

Section 5.2.2, and $RMSE_{min}^{cleaned}$ is the optimal cleaning result obtained under the same parameters without an auto-selected budget.

The strategy's performance is closely tied to the dataset's property, data length, and dirty rate. Table 3 shows that it consistently performs well in most cases, with an average accuracy of 0.915.

Notably, the performance of this strategy can exhibit minor variations in specific scenarios. For instance, Akane's global optimization and pattern sparsity in the short Romanian time series with a trend may lead to over-cleaning of consecutive clean data points, reducing accuracy. As the dirty rate increases, the concentration of dirty points can lead to overestimation due to the reduced frequency of specific patterns. This marginally lowers the strategy's effectiveness on the greedy-based AkaneH approach due to under-cleaning. For 6 of 10 fixed datasets with real dirty data, their dirty rates are pretty low (less than 1%), making it relatively difficult to select the exact budget precisely. However, even in the worst case, this strategy achieves an accuracy of 0.811, and the accuracy is usually more than 0.9. Guided by this strategy, our three methods can consistently outperform the baselines, as detailed in Section 5.2, also emphasizing the strategy's effectiveness.

6 RELATED WORK

6.1 Smoothing-Based Cleaning

Two classic smoothing-based methods are the AR (AutoRegressive) Model [21, 33, 47] and MA (Moving Average) [7], which aim to figure out and eliminate incompatible data points. An AR-type model is used to fit the data, and a complementary criterion, usually the deviation between the prediction and the actual value, is provided to check whether a point is dirty. Through the model fitting, it can learn the normal behavior of data, thus making it capable of identifying and cleaning dirty data. MA calculates the mean (Simple MA) or weighted mean (Weighted or Exponential Weighted MA [18]) of a region centered around a point in the original series and uses it as the cleaned value. MA can obtain good cleaning results by weakening the influence of dirty data.

However, AR neglects the context of data points, while MA alters too much clean data, contravening the minimum change principle.

6.2 Constraint-Based Cleaning

Golab *et al.* [19] proposed the SD (Sequential Dependency) time series data cleaning algorithm. It leverages integrity constraints on attribute dependencies to detect data quality. Song *et al.* [41] further expressed precise speed constraints in the SCREEN algorithm. It offers an idea of concrete maximal and minimal speed within a window and a cleaning method meeting the constraint with minimum change. Later, It has been extended to accommodate multi-speed constraints [16]. These methods can eliminate violations and achieve ideal results with the help of accurate constraints.

Nonetheless, constraints are sometimes hard to obtain or even unavailable. They may make us overlook covert dirty points and lead to cleaning decisions far from the truth, which is improper.

6.3 Statistics-Based Cleaning

Baba *et al.* [4] designed an IR-MHMM (Multi-variate Hidden Markov Model) to address the inherent uncertainty in RFID time series. It obtains the most probable hidden states through training and derives the cleaning results as the most probable observation sequence. Zhang *et al.* [50] incorporate the likelihood-based cleaning concept from relational data cleaning [46] to clean small errors in time series. The core idea involves applying statistics to prioritize low-likelihood data as more likely to be dirty and subsequently correcting it towards higher-likelihood data. Many works [35, 48, 49] have demonstrated the effectiveness of this concept. They suggest the speed changes between consecutive data points are insignificant and more likely to be concentrated around 0. Then, they transform the cleaning problem into finding a sequence having a higher speed change likelihood within the given threshold of series change.

Notably, IR-MHMM targets RFID but not general time series data and SpeedChange focuses on speed change statistics, which is insufficient to make cleaning decisions that are close to the truth.

6.4 Anomaly-Detection-Based Cleaning

Time series anomaly detection is a task aiming to detect anomalies in the time series. Prevalent time series anomaly detection algorithms have been discussed and compared in a recent comprehensive study [37]. To the best of our knowledge, no one has systematically studied whether and how they can be extended to data-cleaning tasks. Some algorithms, like Torsk [20], NumentaHTM [2], TripleES [1], and FFT [34], can take the forecasting or reconstruction series directly as the cleaning result. Other algorithms, like encoding-based GrammarViz [38] or distance-based Sub-LOF [6], can indirectly use the grammar rules or nearest clean neighbors as cleaning suggestions.

However, these algorithms tend to identify windows instead of single points as anomalies, leading to over-cleaning. Besides, dirty data also impedes their models from reliable cleaning decisions.

7 CONCLUSION

In this study, we address the issue of cleaning dirty values in time series data. We begin by highlighting the limitations of dirty data identifications and cleaning decisions of prevalent cleaning algorithms. Recognizing recurrent patterns in the time series and analogizing them to fixed word combinations in textual data, we integrate the concept of perplexity into time series data cleaning. We thus formalize the cleaning problem as minimizing the perplexity of a time series within a given budget and propose a four-phase algorithmic framework. We analyze the impact of dirty data on recurrent patterns and propose an automatic budget selection strategy to ensure feasibility. We make advanced solutions to the framework, including introducing homomorphic pattern aggregation to expand the applicable scenarios and suggesting a heuristic algorithm based on greediness to save resource consumption. These two make the framework more generic. Experiments on 12 real-world datasets show the superiority of our works over 11 existing algorithms.

In the future, we will consider the relationships between different dimensions in multivariate time series and extend the probability calculation phase to compute their perplexity. This will enable us to adapt our framework to multivariate time series. We will also explore the feasibility of utilizing the concept of perplexity to design a new, more potent, and unsupervised time series anomaly detection method.

ACKNOWLEDGMENTS

The authors would like to thank all the anonymous reviewers for their insightful comments and suggestions. This work was supported by the National Key R&D Program of China (No. 2021YFB3300502) and the National Natural Science Foundation of China (No. 92267203).

REFERENCES

[1] Adam Aboode. 2018. Anomaly detection in time series data based on holt-winters method.

[2] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262 (2017), 134–147.

[3] Arthur Asuncion and David Newman. 2007. UCI machine learning repository.

[4] 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 Conference*. ACM, 925–936.

[5] 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 Conference*. ACM, 143–154.

[6] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-Based Local Outliers. In *SIGMOD Conference*. ACM, 93–104.

[7] David R. Brillinger. 2001. *Time series - data analysis and theory*. Classics in applied mathematics, Vol. 36. SIAM.

[8] Márcia R. Cerioli, Luérbio Faria, Talita O. Ferreira, and Fábio Protti. 2011. A note on maximum independent sets and minimum clique partitions in unit disk graphs and penny graphs: complexity and approximation. *RAIRO Theor. Informatics Appl.* 45, 3 (2011), 331–346.

[9] Stanley F Chen, Douglas Beeferman, and Roni Rosenfeld. 1998. Evaluation metrics for language models. (1998).

[10] Wai-Ki Ching and Michael K Ng. 2006. Markov chains. *Models, algorithms and applications* (2006).

[11] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *ICDE*. IEEE Computer Society, 458–469.

[12] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *SIGMOD Conference*. ACM, 1247–1261.

[13] Jerome T. Connor, R. Douglas Martin, and Les E. Atlas. 1994. Recurrent neural networks and robust time series prediction. *IEEE Trans. Neural Networks* 5, 2 (1994), 240–254.

[14] Adrian Dumitrescu and János Pach. 2011. Minimum Clique Partition in Unit Disk Graphs. *Graphs Comb.* 27, 3 (2011), 399–411.

[15] Len Feremans, Vincent Vercruyssen, Boris Cule, Wannes Meert, and Bart Goethals. 2019. Pattern-Based Anomaly Detection in Mixed-Type Time Series. In *ECML/PKDD (1) (Lecture Notes in Computer Science, Vol. 11906)*. Springer, 240–256.

[16] Fei Gao, Shaoxu Song, and Jianmin Wang. 2021. Time Series Data Cleaning under Multi-Speed Constraints. *Int. J. Softw. Informatics* 11, 1 (2021), 29–54.

[17] Ge Gao, Qitong Gao, Xi Yang, Miroslav Pajic, and Min Chi. 2022. A Reinforcement Learning-Informed Pattern Mining Framework for Multivariate Time Series Classification. In *IJCAI*. ijcai.org, 2994–3000.

[18] Everette S Gardner Jr. 2006. Exponential smoothing: The state of the art—Part II. *International journal of forecasting* 22, 4 (2006), 637–666.

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

[20] Niklas Heim and James E. Avery. 2019. Adaptive Anomaly Detection in Chaotic Time Series with a Spatially Aware Echo State Network. *CoRR* abs/1909.01709 (2019).

[21] David J. Hill and Barbara S. Minsker. 2010. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environ. Model. Softw.* 25, 9 (2010), 1014–1022.

[22] Van Long Ho, Nguyen Ho, and Torben Bach Pedersen. 2021. Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information. *Proc. VLDB Endow.* 15, 3 (2021), 673–685.

[23] Van Long Ho, Nguyen Ho, and Torben Bach Pedersen. 2023. Mining Seasonal Temporal Patterns in Time Series. In *ICDE*. IEEE, 2249–2261.

[24] Bryan Hooi, Shenghua Liu, Asim Smailagic, and Christos Faloutsos. 2017. BeatLex: Summarizing and Forecasting Time Series with Patterns. In *ECML/PKDD (2) (Lecture Notes in Computer Science, Vol. 10535)*. Springer, 3–19.

[25] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Söderström. 2018. Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. In *KDD*. ACM, 387–395.

[26] Alexander Ihler, Jon Hutchins, and Padhraic Smyth. 2006. Adaptive event detection with time-varying poisson processes. In *KDD*. ACM, 207–216.

[27] Shawn R. Jeffery, Minos N. Garofalakis, and Michael J. Franklin. 2006. Adaptive Cleaning for RFID Data Streams. In *VLDB*. ACM, 163–174.

[28] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America* 62, S1 (1977), S63–S63.

[29] Dasheng Lee, Chih-Wei Lai, Kuo-Kai Liao, and Jia-Wei Chang. 2021. Artificial intelligence assisted false alarm detection and diagnosis system development for reducing maintenance cost of chillers at the data centre. *Journal of Building Engineering* 36 (2021), 102110.

[30] 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.

[31] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. 2012. Truth Finding on the Deep Web: Is the Problem Solved? *Proc. VLDB Endow.* 6, 2 (2012), 97–108.

[32] Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.* 15, 2 (2007), 107–144.

[33] Kumar G Ranjan, Debesh S Tripathy, B Rajanarayan Prusty, and Debashisha Jena. 2021. An improved sliding window prediction-based outlier detection and correction for volatile time-series. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields* 34, 1 (2021), e2816.

[34] Faraz Rasheed, Peter Peng, Reda Alhajj, and Jon G. Rokne. 2009. Fourier Transform Based Spatial Outlier Mining. In *IDEAL (Lecture Notes in Computer Science, Vol. 5788)*. Springer, 317–324.

[35] 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.

[36] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-Series Anomaly Detection Service at Microsoft. In *KDD*. ACM, 3009–3017.

[37] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. 2022. Anomaly Detection in Time Series: A Comprehensive Evaluation. *Proc. VLDB Endow.* 15, 9 (2022), 1779–1797.

[38] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. 2015. Time series anomaly discovery with grammar-based compression. In *EDBT*. OpenProceedings.org, 481–492.

[39] Shaoxu Song, Chunping Li, and Xiaoquan Zhang. 2015. Turn Waste into Wealth: On Simultaneous Clustering and Cleaning over Dirty Data. In *KDD*. ACM, 1115–1124.

[40] Shaoxu Song and Aoqian Zhang. 2020. IoT Data Quality. In *CIKM*. ACM, 3517–3518.

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

[42] Yunxiang Su, Yikun Gong, and Shaoxu Song. 2023. Time Series Data Validity. *Proc. ACM Manag. Data* 1, 1 (2023), 85:1–85:26.

[43] Yu Sun, Shaoxu Song, Chen Wang, and Jianmin Wang. 2020. Swapping Repair for Misplaced Attribute Values. In *ICDE*. IEEE, 721–732.

[44] Kenneth J. Supowit. 1981. *Topics in Computational Geometry*. Ph. D. Dissertation. University of Illinois Urbana-Champaign, USA.

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

[46] Mohamed Yakout, Laure Berti-Équille, and Ahmed K. Elmagarmid. 2013. Don't be SCAREd: use SCalable Automatic REpairing with maximal likelihood and bounded changes. In *SIGMOD Conference*. ACM, 553–564.

[47] Kenji Yamanishi and Jun'ichi Takeuchi. 2002. A unifying framework for detecting outliers and change points from non-stationary time series data. In *KDD*. ACM, 676–681.

[48] Jie Yang, Alisa Smirnova, Dingqi Yang, Gianluca Demartini, Yuan Lu, and Philippe Cudré-Mauroux. 2019. Scalpel-CD: Leveraging Crowdsourcing and Deep Probabilistic Modeling for Debugging Noisy Training Data. In *WWW*. ACM, 2158–2168.

[49] Zhuoran Yu and Xu Chu. 2019. PIClean: A Probabilistic and Interactive Data Cleaning System. In *SIGMOD Conference*. ACM, 2021–2024.

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

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

Received October 2023; revised January 2024; accepted February 2024