The VLDB Journal
https://doi.org/10.1007/s00778-023-00796-y

REGULAR PAPER

Streaming data cleaning based on speed change

Haoyu Wang[1] · Aoqian Zhang[2] · Shaoxu Song[3] · Jianmin Wang[1]

Received: 20 May 2021 / Revised: 27 September 2022 / Accepted: 9 April 2023

Abstract
Errors are prevalent in data sequences, such as GPS trajectories or sensor readings. Existing methods on cleaning sequential data employ a constraint on value changing speeds and perform constraint-based repairing. While such speed constraints are effective in identifying large spike errors, the small errors that do not deviate much from the truth and indeed satisfy the speed constraints can hardly be identified and repaired. To handle such small errors, in this paper, we propose a cleaning method based on probability of speed change. Rather than declaring a broad constraint of max/min speeds, we model the probability distribution of speed changes. The repairing problem is thus to maximize the probability of the sequence w.r.t. the probability of speed changes. We formalize the probability-based repairing problem and devise algorithms in streaming scenarios. Experiments on real data sets (in various applications) demonstrate the superiority of our proposal.

Keywords Time series · Data cleaning · Stream processing · Speed change

1 Introduction

Data sequences are often found with dirty or imprecise values, such as GPS trajectories or sensor reading sequences [21, 34, 48], which affect downstream applications like classification [30] or traffic prediction [47]. According to the survey [24], even the data of stock prices could be dirty. For instance, the price of SALVEPAR (SY) is misused as the price of SYBASE, which is denoted by SY as well in some sources. In fact, there are two sources of outliers: some outliers come from the errors of measurements, which should be repaired. Others are the part of the signals we want to capture, which should not be modified (see Sect. 7.5 for details).

✉ Shaoxu Song
sxsong@tsinghua.edu.cn

Haoyu Wang
wanghy20@mails.tsinghua.edu.cn

Aoqian Zhang
aoqian.zhang@bit.edu.cn

Jianmin Wang
jimwang@tsinghua.edu.cn

1 School of Software, Tsinghua University, Beijing, China

2 School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China

3 BNRist, School of Software, Tsinghua University, Beijing, China

To clean dirty data, constraint-based repairing is often employed [6]. Existing study [35] on sequential data cleaning considers the constraints on speeds of value changes, namely speed constraints. For example, the speed constraints on fuel meter values state that the fuel consumption of a crane should not be negative and not exceed 40 ls per hour. Constraint-based cleaning identifies the violations to such speed constraints and (minimally) modifies the values so that the repaired results satisfy the speed constraints. According to the minimum change principle, constraint-based cleaning will choose the maximum/minimum allowable values.

An alternative approach is to consider the average of the previous values as a repair, a.k.a. smoothing methods [7, 16]. For example, the simple moving average (SMA) [7] smooths time series data by computing the unweighted mean of the last k points. Instead of weighting equally, the exponentially weighted moving average (EWMA) [16] assigns exponentially decreasing weights over time. As indicated in [35], the problem of the smoothing methods is over-repairing, i.e., almost all the data points are modified, most of which are indeed correct originally and do not need repair.

Moreover, the speed constraints fail to identify the small errors that do not deviate much from the original values, and indeed satisfy the speed constraints. The small errors are particularly important in some applications. For instance, a deviation of 1 m in GPS readings is prevalent and small relative to 10 m large spikes. Such a small error (1 m), however,

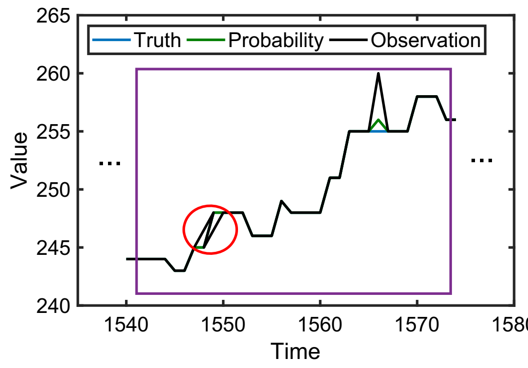

Fig. 1 The challenge example of streaming repairing

is critical in car localization for automatic driving. Moreover, aggregating a large number of small errors, data mining results could be seriously misled, e.g., unable to form meaningful clusters over imprecise GPS readings with many small errors [33]. Our results in Sect. 7.3.3 also show that repairing small errors could improve the accuracy of prediction applications.

Instead of considering the aforesaid max/min speeds, we propose a novel repairing method based on speed change distribution probability in the conference version [48]. However, this version only discusses repairing the sequence as a whole. Actually, in many real scenarios, time sequence comes as a stream. For example, online map apps show the repaired time sequence of GPS trajectory and adjust the recommended routes in real time. Therefore, in this paper, we extend the repairing method to the streaming scenario. Let us first illustrate a challenge example below.

Example 1 Fig. 1 presents a period of stock prices over streaming scenario. (1) The data points after 1574 are unknown for our repairing because they have not come yet. Therefore, in probability distribution construction, we can only use the data points before 1574. (2) The local period from 1541 to 1574 (purple rectangle) is the target of repairing. The data points before 1541 are dropped due to limited memory. Thus, the streaming algorithms should repair locally. (3) An out-of-order arrival is detected at 1548 (red circle). By accident, the data point at time 1548 arrives later than the data point at time 1549. Thus, out-of-order points should be handled by streaming algorithms.

1.1 Challenges

Different from repairing as a whole, there are four main challenges in streaming repairing:

1. Typically, when repairing the entire sequence as a whole, a global optimization problem is considered. However, because infinite data points come continuously in the streaming scenario, each data point should be repaired under a short time latency with limited space. Therefore, algorithms with high time and space complexity are not suitable. It is necessary to look for heuristics, even at the price of accuracy.
2. Budget is a vital parameter when repairing as a whole to avoid over-repairing. However, because of the infinite length of time sequence, it is unrealistic to set the budget in streaming repairing. Hence, we need to find an adaption strategy. Based on it, some repairs will be abandoned even if the probability increases to prevent over-repairing in streaming repairing.
3. Probability distribution can only be constructed with arrived data points, not all data points. Therefore, how to construct a reliable distribution dynamically should be discussed. Meanwhile, a solution of concept shift and drift is also needed.
4. Due to network latency, device failures or data retransmission [25, 35], some of the data points may come later than they should be. Therefore, in general, the time sequence may be out of order. To solve this problem, streaming algorithms should pay special attention to out-of-order points.

1.2 Contributions

Our major contributions are summarized as:

1. We formalize the repairing problem as Problem 1 and Problem 2 based on the idea of maximum probability. Then, we introduce the definition of probability gain and a greedy heuristic in Sect. 4.2.
2. In Sect. 3, we propose the dynamic probability distribution construction. To deal with concept shift and drift, weighted probability construction is introduced in Sect. 3.3.
3. Budget is an important parameter in the repairing problem. In Sect. 5, we present an intuition to select the proper budget when the probability increases much more slowly. Based on its formalization, we introduce several budget-adaptive algorithms.
4. Supported by the idea of probability gain and budget adaptation, we propose several scroll algorithms in Sect. 6.1 and an incremental greedy algorithm in Sect. 6.2 to deal with the dirty and out-of-order stream.

Finally, we report an extensive experimental evaluation in Sect. 7. The results illustrate that our proposal achieves better performance for streaming repairing. Table 1 lists the notations frequently used.

Table 1 Notations

Symbol	Description
x	A sequence of data points
x'	In-order repair of sequence x
$x[i]$ or x_i	Value of i-th data point in x
$x_{i\dots j}$	A subsequence of x from i-th to j-th data points
t_i	Timestamp of i-th data point
θ	Error range of each data point
δ	Cost budget for repairing
u_i	Speed change before and after i-th data point
$\Delta(x, x')$	Repair cost from x to x'
$\log P(x)$	Log probability of a sequence x (Eq. 2)
β	Decay factor for weighted probability construction
$G_i(x_i'')$	Probability gain from repairing x_i' to x_i''
$M^c(x)$	Maximum repaired $\log P(x)$ with $\delta = c$
δ_0	Adaptive budget in adaptive algorithms
$\rho \cdot e^{-\lambda z}$	Exponential function for curve fitting
α	Threshold factor of adaptive budget
w	Sliding window size of adaptive budget
$d_{\max}$	Maximal latency in streaming scenario
W	Window size of streaming algorithms
$\mathcal{G}$	Probability gain set of applied repairs

2 Problem statement

2.1 Preliminaries

Consider a sequence $x = x[1], x[2], \dots$, where each $x[i]$ is the value of the i-th data point from a finite domain.[1] For brevity, we write $x[i]$ as x_i, and $x_{i\dots j}$ denoting the subsequence $x_i, x_{i+1}, \dots x_j$ of x.

Each x_i is associated with a timestamp t_i and an error range θ_i. The error range, e.g., specified by engineering tolerance, denotes that the true value x_i' of i-th data point may be in the range of $[x_i - \theta_i, x_i + \theta_i]$, denoted by $x_i' \in [x_i \pm \theta_i]$. While some data sequences may have individual θ_i for each data point i, e.g., indicated as "accuracy" in GPS readings, others may specify a single $\theta_{\max}$ denoting the maximum error range for all data points in the sequence, such as in sensor readings. Since we do not have any priori knowledge on which points are dirty, every point is potentially a dirty point.

The speed is defined on the *change of value* [35], e.g., $v_{i-1,i} = \frac{x_i - x_{i-1}}{t_i - t_{i-1}}$ from data point $i-1$ to i. Let

$$u_i = v_{i,i+1} - v_{i-1,i} = \frac{x_{i+1} - x_i}{t_{i+1} - t_i} - \frac{x_i - x_{i-1}}{t_i - t_{i-1}} \tag{1}$$

be the *change of speed* before and after the i-th point.

[1] Except QP our methods traverse the repairing space of each data point so that a discretization in advance is needed if the value is continuous.

Converting multiplication to addition by logarithmic operations for calculation facilitation, the log probability $\log P(x)$ of a sequence x is approximated by

$$\log P(x) = \sum_{i=2}^{n-1} \log P(u_i) = \log \prod_{i=2}^{n-1} P(u_i) \tag{2}$$

where $P(u_i)$ denotes the probability of speed change u_i.

Equation 2 makes a strong simplification that speed changes are independent. We note that this type of simplification may hardly be justified in practice. Unfortunately, even with such a simplification, the repairing problem is already NP-hard, as shown in Sect. 2.3. Therefore, we have to make a trade-off between repairing overhead and accuracy. Nevertheless, although such a simplification may be violated, such as in financial time series, the results over STOCK (corresponding to Fig. 1) show that our algorithms can still perform well under the simplification, in Sect. 7.2.

Assumption 1 Errors occur randomly both in terms of occurrence likelihood and in terms of values.

In other words, errors may occur at any point i in the time series. Moreover, the erroneous value x_i could be any value in the error range $[x_i' - \theta_i, x_i' + \theta_i]$, or equivalently $x_i' \in [x_i \pm \theta_i]$ as introduced at the beginning of Sect. 2.1.

Assumption 2 Errors are infrequent in the dirty time sequence.

This assumption is obeyed by many datasets. For example, only 7% of data in a stock dataset is inaccurate [35] and real dataset GPS has only 6.4% of dirty points. Meanwhile, similar assumptions are shown in other outlier detection and repairing works [27, 46]. Without prior knowledge, all these works including ours learn expected patterns from the dirty sequence and too many errors inevitably distort the patterns and repairs [5].

Assumption 3 The sequence of speed changes is stationary for a period of time.

Differencing [20] is a widely used technology to make a non-stationary time sequence stationary. As indicated in [20], it is almost never necessary to use more than second-order differencing in practice. In fact, speed change can be regarded as a two-order differencing so that the sequence of speed changes is nearly stationary. For example, the p-values of augmented Dickey-Fuller test [9] on the speed changes of STOCK and GPS are both less than 0.001 showing the stationarity. Although speed change sequence with seasonal pattern is not suitable, there are still many datasets obeying this assumption.

Based on Assumptions 2 and 3, how to construct the empirical probability distribution on speed changes is discussed in Sect. 3.

2.2 Problem of repairing as a whole

For repairing as a whole, we consider a finite sequence. Following the same line of maximal probability repairing over relational data [44], we propose the repair problem over sequential data as follows.

Problem 1 Given a finite sequence x of n data points and a repair cost budget δ, the *maximum probability problem for repairing as a whole* is to find a repair x' such that $\Delta(x, x') \leq \delta$ and $\log P(x')$ is maximized.

Notably, if we put no limitation on the repair, the speed change sequence may be smoothed to one that is constantly equal to the mode of the data. To avoid it, referring to the minimum change principle in data repairing [6], we consider the repair cost from x to x' as in [35] and give a budget δ for it:

$$\Delta(x, x') = \sum_{i=1}^{n} |x'_i - x_i| \leq \delta$$

As illustrated in Sect. 5, such a budget could be determined adaptively.

The dirtiness is thus not evaluated in terms of deviation from the mode, but the deviation from the probability distribution. If a speed change is frequently observed, its probability is high and thus not likely to be regarded as dirty. On the contrary, a rare speed change is more likely to be caused by a dirty value due to its low probability.

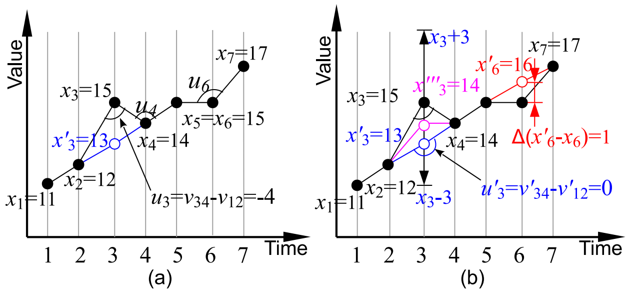

Fig. 2 Possible repairs of an example sequence

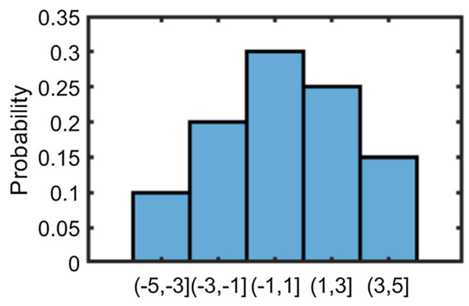

Fig. 3 Example speed change probability distribution

We formalize the repair problem as follows.

$$\begin{aligned} \max \quad & \sum_{i=2}^{n-1} \log P\left(\frac{x'_{i+1} - x'_i}{t_{i+1} - t_i} - \frac{x'_i - x'_{i-1}}{t_i - t_{i-1}}\right) \\ \text{s.t.} \quad & \sum_{i=1}^{n} |x'_i - x_i| \leq \delta \\ & x'_i \in [x_i \pm \theta_i] \qquad 1 \leq i \leq n \end{aligned} \tag{3}$$

With the limit of δ and θ, repair will not change data distribution too much compared to the original data. Thereby, we construct the probability distribution on the original data as shown in Sect. 3.1, without updating the data distribution P as data repairing.

Example 2 Let's consider a sequence x ={11, 12, 15, 14, 15, 15, 17}, with timestamps t = {1, 2, 3, 4, 5, 6, 7}. Figure 2a illustrates the data points (in black), and Fig. 3 shows the corresponding probability distribution of speed changes.

The probability of speed change on the 3rd point (x_3) is

$$P(u_3) = P\left(\frac{14-15}{4-3} - \frac{15-12}{3-2}\right) = P(-4) = 0.1,$$

with $\log P(u_3) = \log(0.1) = -2.3$. By similarly computing the probability on other data points, we have the log probability of x, i.e., $\log P(x) = \log(0.25) + \log(0.1) + \log(0.25) + \log(0.2) + \log(0.25) = -8.1$.

Since the third observation has $\log P(u_3) = -2.3$, much lower than those of all others like $\log P(u_2) = -1.4$ or $\log P(u_4) = -1.6$, we may identify it as dirty data with a lower probability of speed change. Such a dirty value may be introduced by various reasons, such as measurement error or sensor failures. The repairing thus proposes to modify the third observation for a larger speed change probability.

The maximum (log) probability will be reached when the probability of speed change $P(u_i)$ is maximized for each u_i, i.e., having speed change $u_i \in (-1, 1]$ with $P(u_i) = 0.3$. Consequently, the maximum log probability is $\log P(x^*) = 5 * \log(0.3) \approx -6.0$.

2.3 Hardness

Consider the sequence $x = \{11, 12, 15, 14, 15, 15, 17\}$ in Example 2. Suppose that the error range is $\theta_i = 3$ for all the data points i. That is, for each point x_i, there are 7 potential modifications, $x'_i = \{x_i - 3, \dots, x_i + 3\}$. A large number of 7^7 combinations could be considered as possible repairs. In particular, the repairing of x'_i is affected by the choices of x'_{i-1} and x'_{i+1} w.r.t. the speed change probability (in Fig. 3). Intuitively, we can build a reduction from the 0/1 knapsack problem, by modeling the item values as the speed change probabilities, and thus show the hardness of our repair problem above.

Theorem 1 *Given a sequence x with error range θ, repair cost budget δ, and threshold ℓ, the problem is* NP*-complete to determine whether exists a repair x' of x such that $\Delta(x, x') \leq \delta$ and $\log P(x') \geq \ell$.*

Proof The problem is clearly in NP. Given a repair x', it can be verified in polynomial time whether each point repair x'_i is in the valid range and $\Delta(x, x') \leq \delta$. Besides, $\log P(x')$ can also be computed in polynomial time.

To prove the NP-hardness, we show a reduction from the 0/1 knapsack problem, which is one of Karp's 21 NP-complete problems [22]. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity, the problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less than or equal to the knapsack's capacity.

We create five data points with values $x_{i1}, \dots, x_{i5}$ for each item i, having

$$
\begin{aligned}
x_{i1} &= b_i, & \theta_{i1} &= 0,\\
x_{i2} &= b_i, & \theta_{i2} &= 0,\\
x_{i3} &= 2b_i, & \theta_{i3} &= w_i,\\
x_{i4} &= 4b_i, & \theta_{i4} &= 0,\\
x_{i5} &= 7b_i, & \theta_{i5} &= 0,
\end{aligned}
$$

where $b_i = 4 * (w_1 + \dots + w_{i-1}) + 2w_i + i$.

The log probabilities of speed changes are defined[2] as

$$
\log P(u_i) = \begin{cases} v_i/3 & \text{if } u_i = u_{i1} = b_i - w_i,\\ v_i/3 & \text{if } u_i = u_{i2} = b_i + w_i,\\ v_i/3 & \text{if } u_i = u_{i3} = b_i - 2w_i\\ v_i/3 & \text{if } u_i = u_{i4} = b_i + 2w_i,\\ 0 & \text{otherwise} \end{cases}
$$

for $i = 1, \dots, n$.

We can show that there is a subset of items with total weight W and total value V, if and only if there is a repair x' with $\Delta(x, x') = W$ and $\log P(x') = V$. □

2.4 Streaming repair problem

In a stream, data points may come out of order. For example, in the Internet of Things (IoT) scenario, sensors collect data and transmit them to servers, where complex operations like data repairing are conducted. Due to network latency, device failures or data retransmission [25, 35], delay may occur to some of the data points. In other words, servers may receive them later than the data points with larger timestamps, leading to the out-of-order time sequence. When merging data from independent streams with clocks that cannot be synced, out-of-arrivals may also occur. However, in such a scenario, the timestamps by unsynchronized clocks are also dirty and need repair [14], which is out of the scope of this study.

Note that we construct the probability distribution based on the statistics of the currently arrived data points in the stream. Our study thus focuses on the repairing problem in the presence of out-of-order arrivals. It will be more challenging to consider out-of-order arrivals in both the probability distribution construction and data repairing. Intuitively, if one of the sensor sources transmits slower than it should, unlike random occurrences, there are some patterns of delay that can be utilized in the probability distribution construction. If the out-of-order arrivals occur in batches, the probability distribution constructed on the currently arrived data points may not be reliable for repairing. We leave the interesting yet challenging direction for future study.

Suppose $r(x_i)$ is the index of x_i after reordering the dirty sequence x, the repair cost from $x_{i\dots j}$ to $x'_{r(x_{i\dots j})}$ is

$$
\Delta(x_{i\dots j}, x'_{r(x_{i\dots j})}) = \sum_{k=i}^{j} |x'_{r(x_k)} - x_k|
$$

[2] The log probability is only defined in this proof. In actual repairing, it is constructed from original data.

Meanwhile, only a period of data points can be repaired at one time, which corresponds to the scroll or sliding windows in Sect. 6. The number of relevance in repairing is not the total number of data points in the probability distribution construction but the length of the window. The length could be occasionally 3, or any other user-specified, such as 730. The experimental results in Fig. 26, 27 and 28 in Sect. 7.4.2 show the trade-off by the size of window. That is, a larger window size leads to better repair RMS error with more local data points considered but higher time cost. A smaller window size is the opposite. Besides, it is impractical to manually specify the dynamic budget δ for each period.

Considering above difficulties, the streaming repair problem over sequential data is as follows.

Problem 2 Given a period $x_{i\dots j}$ in an out-of-order stream x, the *streaming maximum probability repair* problem is to find an in-order repair $x'_{r(x_{i\dots j})}$ such that $\Delta(x_{i\dots j}, x'_{r(x_{i\dots j})}) \leq \delta(x_{i\dots j})$ and $\log P(x'_{i\dots j})$ is maximized.

We formalize the repair problem as follows.

$$
\begin{aligned}
\max \quad & \sum_{k=i+1}^{j-1} \log P\left(\frac{x'_{k+1}-x'_k}{t_{k+1}-t_k} - \frac{x'_k - x'_{k-1}}{t_k - t_{k-1}}\right) \\
\text{s.t.} \quad & \sum_{k=i}^{j} |x'_{r(x_k)} - x_k| \leq \delta(x_{i\dots j}) \\
& x'_{r(x_k)} \in [x_k \pm \theta_k] \qquad\qquad i \leq k \leq j
\end{aligned}
\tag{4}
$$

Example 3 Consider the out-of-order sequence $x = \{11, 12, 14, 15, 15, 17, 15\}$, with timestamps $t = \{1, 2, 4, 3, 5, 7, 6\}$. When the first three data points arrive, we simply get $u = \frac{14-12}{4-2} - \frac{12-11}{2-1} = 0$ and put it into the probability distribution. After that, when delayed point x_4 arrived, it is inserted into the correct place. Thus, $u = \frac{14-15}{4-3} - \frac{15-12}{3-2} = -4$ is put. Following the same line, we construct the probability distribution gradually. The details of probability construction are shown in Sect. 3.

For repairing, x_4 and x_7 are recognized as dirty, and repaired to 13 and 16, respectively, using the scroll window algorithms in Sect. 6.1. Besides modifying the dirty data points, we also reorder according to the timestamps. Thus, the repaired sequence is $x = \{11, 12, 13, 14, 15, 16, 17\}$, with timestamps $t = \{1, 2, 3, 4, 5, 6, 7\}$.

2.5 Normalization

In the scenario of repairing the time sequence as a whole, the entire time series is given as the input, together with its budget δ. The DPL method (presented in the conference version [48]) introduces a normalization factor H and maps

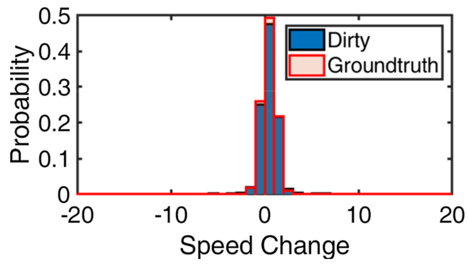

Fig. 4 The comparison between speed change distribution constructed from original dirty sequence and the ground truth over GPS dataset

the budget from δ to $\frac{\delta}{H}$. It greatly reduces the candidate values and thus improves the time efficiency of repairing.

In the streaming scenario, the budget is not fixed and thus cannot apply the normalization in DPL. Intuitively, the values are within the error range w.r.t. θ as discussed after the aforesaid Assumption 1. In this sense, normalization on θ will not encounter data that are out of bounds. Similar to the normalization on δ, we introduce the normalization factor H on θ. In other words, we only consider candidates in $x'_i \in \{x_i \pm (H \cdot j) \mid j \in [-\lfloor\frac{\theta_i}{H}\rfloor, \lfloor\frac{\theta_i}{H}\rfloor]\}$ rather than the entire range. Since the normalization does not change the error range, reducing only the candidates, the definition of outliers and the computation of distribution are not affected.

3 Probability distribution construction

3.1 Construction on dirty sequence

Ideally, the speed change probability should be constructed from the ground truth to obtain real distribution, which is often unlikely in practice. Therefore, we propose to construct on the original (dirty) data. According to Assumption 3, we use the frequency of the speed change as the approximation of the probability. The probability construction is independent of data repairing. Specifically, when a data point is modified in repairing, the probability distribution $\hat{P}$ should not be modified as well.

It is true that the frequency distribution works well only if the data are discrete and if each category has a sufficiently high probability. Note that our distribution is about the speed changes, rather than the original values possibly with high precision. For example, for the financial data such as in Fig. 1, rather than studying the frequency distribution of the original STOCK prices, we construct the probability distribution of speed changes on the STOCK prices. To avoid isolated bars, binning is used to construct the distribution, i.e., nearby speed changes are gathered in the same bin. With proper bin width on speed changes, even if the original data is high precision, the difference in the repaired data is not significant.

The experimental results in Fig. 15 in Sect. 7.2.4 show that the RMS error of the repair is low over the STOCK data.

Referring to the methods in [11], we use JS-divergence to describe the *statistic distortion*. Due to Assumption 2, the distortion is not severe. For example, the JS-divergence of two distributions in Fig. 4 over GPS dataset is only 0.0012, implying that $\hat{P}$ is an opposite approximation of real distribution. Moreover, we can clean the probability distribution in advance if necessary. Common cleaning strategies, such as K-Sigma principles and outlier detection, can be used in the cleaning.

3.2 Simple distribution construction

In a stream, we use all of the currently arrived data points in the probability distribution construction. When new data points arrive in the stream, we update the distribution referring to the new arrivals.

To trust the distribution estimator, we need adequately many instances. Therefore, we empirically show the number of data points that are sufficient to obtain statistical distribution steady in a data stream. For example, Fig. 5 illustrates the statistical distributions over the first 100, 1000 and 10000 data points, respectively. As shown, the distribution over 1000 points is already very close to the one over a significantly larger 10000 points.

To further illustrate the number of data points that are sufficient for statistical distribution in practice, Fig. 6 shows the JS-divergence between the statistical distributions over the first m data points in a time series and all the n data points in the whole time series. As shown in different datasets, by using the first 500 data points, the statistical distribution is already very close to the one over the entire dataset.

Nevertheless, we conduct a statistical significance test, two-sample Kolmogorov–Smirnov test [3]. The null hypothesis is that the distributions constructed with the first m data points and all n data points are from the same distribution. It is rejected when the p-value is smaller than the significance level, e.g., 0.05. As shown in Table 2, the hypothesis with various m in all three datasets is not rejected, illustrating that the distributions are close.

Finally, the experiments in Figs. 29, 30 and 31 in Sect. 7.4.3 show that by even using only the first 200 arrived data points, the constructed distribution already leads to similar repair performance compared with those built on 500 arrived data points, in various datasets. In this sense, the distributions obtained from the first hundreds of points are already as reliable as the computations over more remaining points.

3.3 Weighted distribution construction

Concept shift and drift [12, 15, 38] are common characteristics of stream data. In this paper, it means the speed change distribution may be changed over time. Thus, simple probability construction in Sect. 3.2 which uses all arrived data points equally in distribution is not suitable.

To deal with it, we propose a weighted probability construction with a forgetting mechanism [19]. Suppose that there are n arrived data points currently. We assign $e^{\beta(i-n)}$ weight to speed change u_i, where β is a decay factor. Thus, with exponential weight, the close data point will play a much more important role. When $\beta = 0$, it degenerates to simple construction with equal weight.

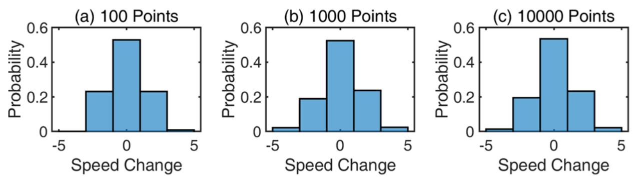

Fig. 5 Speed change distribution constructed with various data points over HHAR

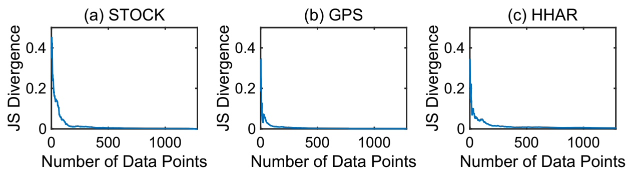

Fig. 6 JS-divergence between the distribution constructed on various number of data points and the distribution on all the data points in the dataset

Table 2 Results of two-sample KS tests

m	STOCK		GPS		HHAR	
	Rej	P-value	Rej	P-value	Rej	P-value
100	No	0.60	No	0.51	No	1.00
200	No	0.97	No	0.58	No	0.89
300	No	0.98	No	0.54	No	0.72
500	No	1.00	No	0.80	No	0.67
700	No	1.00	No	0.81	No	0.87
1000	No	1.00	No	0.59	No	0.75

3.4 Clarification on distribution assumption

It is worth noting that only the QP and SG algorithms proposed in the conference version [48] need the assumption of Gaussian distribution, used for more efficient computation. All the methods studied in this study do not need such an assumption. For instance, Fig. 11 illustrates the statistical distributions of six different datasets, some of which are obviously not Gaussian distribution.

Instead, what we assume is that the statistical distributions, either Gaussian or not, are steady in different parts of the time series, as discussed in Assumption 3. In this sense, the statistical distribution, either on the whole dataset for Repairing as a Whole or on the first part of the time series for Streaming Repair, can be utilized to advise the repair in different parts of the data. In particular, for the streaming scenario, the statistical distribution that is close to the one of the whole dataset can be followed. Again, as illustrated in Fig. 6, it is often sufficient to consider the statistical distribution over the first 500 data points in the dataset.

4 Repair in a window as a whole

In this section, we discuss repairing the subsequence in an independent window. Because the subsequence can be seen as a whole, it is a repair problem as Problem 1. Referring to the conference version [48], the dynamic programming algorithms are introduced briefly. After that, a universal greedy algorithm is introduced for efficiency.

In this scenario, all data points in the window are accessible. Before repairing, we can easily reorder the data points according to their timestamps. Thus, out-of-order arrival is not considered in this section.

4.1 Dynamic programming algorithms

Based on the idea of dynamic programming, we divide the entire problem into a series of sub-problems. Let $x'_{1\ldots i}$ be a repair of the subsequence $x_{1\ldots i}$ with the maximum log probability[3] $\log P(x'_{1\ldots i})$, whose cost is $\Delta(x'_{1\ldots i}, x_{1\ldots i}) = c_i$, and the last two values of $x'_{1\ldots i}$ are x'_{i-1}, x'_i, respectively. We denote this maximum log probability $\log P(x'_{1\ldots i})$ by $D(i, c_i, x'_{i-1}, x'_i)$.

The recurrence computation is as follows

$$\begin{aligned} &D(i, c_i, x'_{i-1}, x'_i) \\ &\quad = \max_{x'_{i-2}\in[x_{i-2}\pm\theta_{i-2}]} D(i-1, c_{i-1}, x'_{i-2}, x'_{i-1}) + \log P(u'_{i-1}) \end{aligned} \tag{5}$$

where $c_{i-1} = c_i - \Delta(x'_i, x_i)$, and $u'_{i-1} = \frac{x'_i - x'_{i-1}}{t_i - t_{i-1}} - \frac{x'_{i-1} - x'_{i-2}}{t_{i-1} - t_{i-2}}$.

Initially, for $i = 2$, we have

$$D(2, c_2, x'_1, x'_2) = 0, \forall x'_1 \in [x_1 \pm \theta_1], \forall x'_2 \in [x_2 \pm \theta_2].$$

This algorithm is called DP with $O(n\theta_{\max}^3\delta)$ time complexity and $O(n\theta_{\max}^2\delta)$ space complexity.

Besides, DPC is the dual algorithm of DP. Let $x'_{1\ldots i}$ be a repair of the subsequence $x_{1\ldots i}$, whose log probability is $\log P(x'_{1\ldots i}) = l_i$, the last two values of $x'_{1\ldots i}$ are x'_{i-1}, x'_i, respectively, and the cost $\Delta(x'_{1\ldots i}, x_{1\ldots i})$ minimized. We denote this minimized cost $\Delta(x'_{1\ldots i}, x_{1\ldots i})$ by $C(i, l, x'_{i-1}, x'_i)$.

The recurrence computation is as follows

$$\begin{aligned} &C(i, l_i, x'_{i-1}, x'_i) \\ &\quad = \min_{x'_{i-2}\in[x_{i-2}\pm\theta_{i-2}]} C(i-1, l_{i-1}, x'_{i-2}, x'_{i-1}) + \Delta(x'_i, x_i) \end{aligned} \tag{6}$$

where $l_{i-1} = l_i - \log P(u'_{i-1})$, and $u'_{i-1} = \frac{x'_i - x'_{i-1}}{t_i - t_{i-1}} - \frac{x'_{i-1} - x'_{i-2}}{t_{i-1} - t_{i-2}}$.

Initially, for $i = 2$, we have

$$C(2, 0, x'_1, x'_2) = \Delta(x'_1, x_1) + \Delta(x'_2, x_2),$$

$$\forall x'_1 \in [x_1 \pm \theta_1], \forall x'_2 \in [x_2 \pm \theta_2].$$

DPC is a quadratic-time (for fixed error range θ), constant-factor approximation algorithm, which runs in $O(n^2\theta_{\max}^3)$ time with $O(n^2\theta_{\max}^2)$ space.

4.2 Universal greedy algorithm

Low time efficiency is an essential problem of dynamic programming algorithms. Thus, we introduce a universal greedy algorithm which computes much faster. Moreover, this *universal* algorithm has no requirements on the distribution of speed changes (different from SG [48] assuming Gaussian distribution). Besides, this idea is also adapted to the incremental greedy algorithm in Sect. 6.2 to support stream computation.

4.2.1 Probability gain

Firstly, we talk about the probability gain of a repair. Let x' be the current repair, initially $x' = x$. Suppose x''_i is a repair of a single data point x'_i. When this repair is applied, u_{i-1}, u_i and u_{i+1} will be changed. According to Eq. 2, if x'_i is modified to x^*_i, the influenced terms in $\log P(x')$ is as follows.

$$\begin{aligned} Q_i(x^*_i) = {} & \log P\left(\frac{x^*_i - x'_{i-1}}{t_i - t_{i-1}} - \frac{x'_{i-1} - x'_{i-2}}{t_{i-1} - t_{i-2}}\right) \\ & + \log P\left(\frac{x'_{i+1} - x^*_i}{t_{i+1} - t_i} - \frac{x^*_i - x'_{i-1}}{t_i - t_{i-1}}\right) \\ & + \log P\left(\frac{x'_{i+2} - x'_{i+1}}{t_{i+2} - t_{i+1}} - \frac{x'_{i+1} - x^*_i}{t_{i+1} - t_i}\right) \end{aligned} \tag{7}$$

In our greedy strategy, the repair cost should be increased by every repair. Thus, probability gain of repair x''_i is defined as

$$G_i(x''_i) = \begin{cases} Q_i(x''_i) - Q_i(x'_i) & , \Delta(x''_i, x_i) > \Delta(x'_i, x_i) \\ 0 & , \Delta(x''_i, x_i) \le \Delta(x'_i, x_i) \end{cases} \tag{8}$$

[3] Probability distribution is constructed before dynamic programming and remains unchanged during the repairing.

where $Q(i, x_i'')$ and $Q(i, x_i')$ are the log probability after and before this new repair, $\Delta(x_i'', x_i)$ and $\Delta(x_i', x_i)$ are the repair cost, respectively. Obviously, only when a repair x_i'' increases both the probability and the cost, we have $G_i(x_i'') > 0$.

4.2.2 Universal greedy

Based on the above concept, the greedy idea is to continuously apply the repair with the highest probability gain. In the beginning, we initialize the table of probability gain. For each $i \in \{1, 2, \dots n\}$ and $x_i'' \in [x_i \pm \theta_i]$, we compute $G_i(x_i'')$ and keep them with a max-heap.

In each iteration, we choose the repair x_i'' on the top of the heap. If the cost of x_i'' is not larger than the remaining budget, just apply it. Otherwise, remove it from the heap and continue choosing the top until a cost-proper repair. Due to $\delta \gg \theta_i$ in most real datasets, only the last several repairs will suffer from the insufficient budget. After choosing and applying, we recalculate the following gains and adjust the heap.

$$G_j(x_j''),\ j \in \{i-2, i-1, i, i+1, i+2\},\ x_j'' \in [x_j \pm \theta_j]$$

The algorithm terminates after budget δ is used up or no $G_i(x_i'')$ is above zero. Algorithm 1 presents the main procedure of this universal greedy algorithm.

```
Algorithm 1: UG($x, \theta, \delta$)
   Data: data sequence $x$, error range $\theta$, repair budget $\delta$
   Result: the repaired data sequence $x'$
1  $x' \leftarrow x$;
2  $cost \leftarrow \delta$;
3  initialize $G_i(x_i'')$ for each $i, x_i''$ and put into $heap$;
4  while $cost > 0$ do
5      $G_i(x_i'') \leftarrow heap.top()$;
6      $heap.removeTop()$;
7      if $G_i(x_i'') \leq 0$ then
8          break;
9      if $cost \geq \Delta(x_i'', x_i) - \Delta(x_i', x_i)$ then
10         $x_i' \leftarrow x_i''$;
11         $cost \leftarrow cost - (\Delta(x_i'', x_i) - \Delta(x_i', x_i))$;
12         modify $G_j(x_j''), j \in [i \pm 2], x_j'' \in [x_j \pm \theta_j]$ and adjust
           $heap$;
13 return $x'$
```

Example 4 **(Universal Greedy Algorithm)** Consider the sequence x ={11, 12, 15, 14, 15, 15, 17} in Example 2 with the probability distribution in Fig. 3. Suppose $\theta_i = 2$. At the beginning, we calculate $G_i(x_i')$ for each i and x_i', then keep them in a max-heap.

In the first iteration, the heap is shown in Table 3. $G_3(13) = 1.5$ is on the top. Therefore, we apply the repair, i.e., $x_3' = 13$, with the cost of 2 and adjust the heap.

Table 3 Example of universal greedy

Element	Iteration 1	Iteration 2
1	$x_3' = 13, G = 1.5$	$x_6'' = 16, G = 0.6$
2	$x_3' = 14, G = 1.1$	$x_6'' = 17, G = 0.2$
3	$x_4' = 16, G = 0.9$	$x_5'' = 14, G = 0.2$
4	$x_5' = 14, G = 0.8$	$x_7'' = 15, G = 0.2$
5	$x_5' = 13, G = 0.6$	$x_7'' = 16, G = 0.2$
6	$\dots$	

In the second iteration, the greedy algorithm chooses the top $G_6(16) = 0.6$, generating the new repair $x_6'' = 16$. Then, the probability cannot be further increased (The heap is empty). Thus, the algorithm is terminated. Finally, the optimal sequence is {11, 12, 13, 14, 15, 16, 17} with $\delta \geq 3$.

If the budget is not enough, e.g., $\delta = 2$, the algorithm will stop after one iteration and return {11, 12, 13, 14, 15, 15, 17}.

Proposition 1 *The greedy algorithm runs in $O((n + \delta)\theta_{\max} \log(n\theta_{\max}))$ time.*

Proof First, we should calculate $G_i(x_i'')$ for each $i \in [1, n]$ and $x_i'' \in [x_i \pm \theta_i]$. Thus, there are $O(n\theta_{\max})$ different elements. Building a heap with such elements needs $O(n\theta_{\max})$ time.

Then, for each iteration, the time of selecting the top element and applying is $O(1)$. After that, $O(\theta_{\max})$ elements should be adjusted, whose time complexity is $O(\theta_{\max} \log (n\theta_{\max}))$. There are at most δ iterations.

Finally, UG costs $O((n + \delta \log(n\theta_{\max}))\theta_{\max})$ time. □

5 Budget determination

The repair cost budget δ is a pivotal parameter for repairing highly related to the repair accuracy. If δ is too small, errors in the sequence cannot be cleaned completely. On the contrary, over-repairing happens when δ is too large. Therefore, an automatic budget determination can not only reduce the burden to specify parameters but also prevent serious performance loss caused by improper budget.

In this section, we introduce an intuition of determining δ. After that, we propose a detailed mathematical method to find a proper δ. Finally, we improve the above algorithms to make them budget-adaptive.

5.1 Intuition

Through a large number of experiments on multiple datasets, we have a practical intuition to set budget δ properly: δ_0 is selected as the proper one if the probability increases much

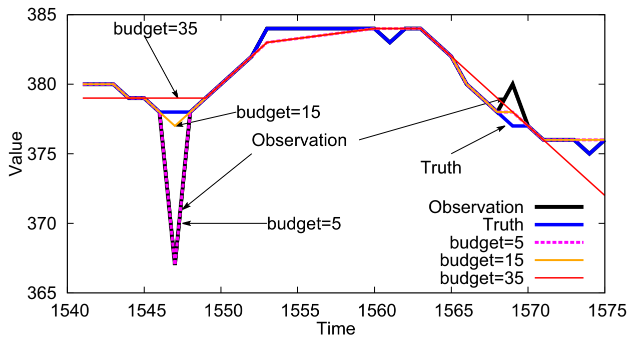

Fig. 7 Example repairs with various repair cost budgets δ

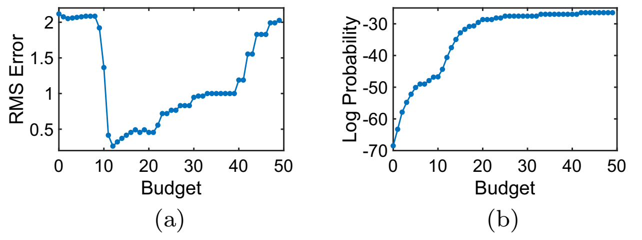

Fig. 8 Repair result **a** error and **b** log probability under various repair cost budgets δ

more slowly when $\delta > \delta_0$. The following example explains it in detail.

Example 5 **(Repair and cost)** Consider the sequence in Fig. 7, a real segment from the STOCK[4] dataset. Given a small repair cost budget, e.g., $\delta = 5$, referring to the optimization problem in Eq. 3, a repair x' will be returned, with four points changed at time 1553, 1561, 1569 and 1573, as shown in Fig. 7. It has repair cost $\Delta(x, x') = |x_{1553} - x'_{1553}| + |x_{1561} - x'_{1561}| + |x_{1569} - x'_{1569}| + |x_{1573} - x'_{1573}| = 5 \leq \delta$, with the maximized log probability $\log P(x') = -50.1$. Notably, the large spike at time 1547 could not be repaired under this small budget.

On the other hand, if the repair cost budget is too large, e.g., $\delta = 35$, a repair x' with a large number of modified points is returned. The corresponding repair cost is $\Delta(x, x') = 33 \leq \delta$, with the maximized log probability $\log P(x') = -27.0$.

Intuitively, we would consider a "proper" setting of repair cost budget, e.g., $\delta = 15$ with $\log P(x') = -32.8$ which is neither too small ($\delta = 5$ with data points barely repaired and low returned probability) nor too large ($\delta = 35$ with data points over repaired but no large probability gain). Following this guideline, a good δ could be practically chosen by observing the log probabilities returned together with the repair results. For instance, the probability does not greatly increase by setting δ from 15 to 20 in Fig. 8b. That is, with δ in the range from 15 to 20, the data are neither insufficiently repaired nor over-repaired.

[4] http://finance.yahoo.com/q/hp?s=AIP.L+Historical+Prices.

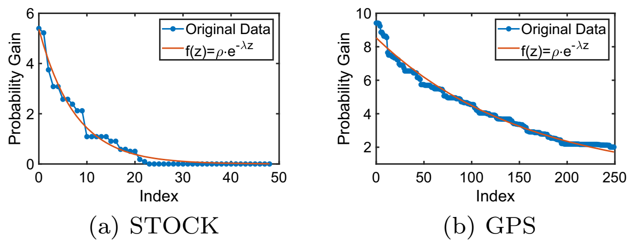

Fig. 9 Probability gain in descending order and its fitting curve

5.2 Formalization

The relationship between probability gain and budget indicates an evident long tail as shown in Fig 8. According to Pareto Rule [1], the top 20% budget achieves 80% probability gain, which points out the transition point in the long tail distribution. Referring to the intuition above, δ_0 should achieve 80% of the maximum probability gain.

Example 6 Consider the sequence in Example 5, its log probability varies with the budget δ as shown in Fig. 8b. Without repairing, the log probability is $\log P(x) = -68.5$ while the maximum log probability with $\delta = 50$ is $\log P(x') = -26.5$. Thus, the log probability with 80% gain is $-68.5 + 0.8 * [(-26.5) - (-68.5)] = -34.9$. $\delta = 15$ is the minimum budget whose corresponding log probability is above -34.9, which is regarded as the proper one.

Calculating the maximum probability is computation-consuming. In the worst case, the probability with $\delta = \sum_{i=1}^{n} \theta_i$ should be calculated, i.e., all data points are repaired with the maximum range. Therefore, we hope to discover the pattern of probability, thereby reducing complexity.

Example 7 Consider again the log probability with various δ in Fig. 8b. We calculate the probability gain and sort them in the descending order, as shown in Fig. 9a, together with the fitting curve $f(z) = \rho \cdot e^{-\lambda z} = 5.369e^{-0.1319z}$. With the coefficient of determination $R^2 = 0.9799$, we believe that the probability gain follows this exponential expression. Similar results are also shown in Fig. 9b with $R^2 = 0.9833$.

Therefore, instead of picking up the budget with 80% maximum probability gain, we calculate the probability gain with a smaller δ and fit with an exponential curve.

Proposition 2 *Let $f(z) = \rho \cdot e^{-\lambda z}$ with $\lambda > 0$, we have*

$$\int_0^m f(z)dz = a \int_0^{+\infty} f(z)dz \;\Leftrightarrow\; f(m) = (1-a)\rho$$

Proof $F(z) = -\frac{\rho}{\lambda} e^{-\lambda z}$ is the indefinite integral of $f(z)$. Thus, we have $-\frac{\rho}{\lambda}(e^{-\lambda m} - 1) = a\frac{\rho}{\lambda}$, which leads to $f(m) = \rho e^{-\lambda m} = (1-a)\rho$. □

After getting the parameters ρ and λ, we use the following equation to select the proper budget:

$$\delta_0 = \min\left\{\delta \middle| \frac{M^{\delta+w/2}(x) - M^{\delta-w/2}(x)}{w} < \alpha\rho\right\} \tag{9}$$

where $M^{\delta}(x)$ is the maximum log probability of sequence x with budget δ and threshold factor $\alpha = 0.2$. To smooth the influence of noises, we use a sliding window whose size is w.

5.3 Budget-adaptive algorithms

With Eq. 9, algorithms can be budget-adaptive, i.e., automatically terminated within the proper budget. In this part, the budget-adaptive versions of DP, DPC and UG are introduced. Their complexities are the same as the original ones.

5.3.1 Adaptive-DP

For the dynamic programming algorithm DP, suppose the exponential fitting parameter ρ is estimated, a naive way to achieve budget-adaptive is running DP with increasing δ. Then, calculate the probability of sequence for each δ and terminate according to Eq. 9. However, there is too much overlapping computation. The time complexity degenerates to $O(n\delta_0^2\theta_{\max}^3)$ where δ_0 is the adaptive budget. To reduce the complexity, we should compute incrementally under different budgets. The main procedure is presented in Algorithm 2.

In the outermost loop, we continuously increase c. In every iteration, we have

$$M^c(x) = \max_{x'_{n-1}, x'_n} D(n, c, x'_{n-1}, x'_n).$$

The outermost loop terminates until δ_0 is found.

Now, the only problem is when to fit and estimate ρ. It is divided into two stages: The first stage is rough estimation to determine when for curve fitting. Fitting with too much data leads to a waste of computation while the contrary leads to inaccurate parameter estimation. We estimate ρ with $f(0)$, i.e., the maximum probability gain. When the following condition is met with the current cost c, we go to the second stage:

$$\frac{M^c(x) - M^{c-w}(x)}{w} < \alpha \cdot \max_{w \le i \le c}\left\{\frac{M^i(x) - M^{i-w}(x)}{w}\right\}$$

The second stage is a fine estimation. We make the exponent fitting with the existing probability gains, i.e., $M^i(x), 0 \le i \le c$, to estimate ρ. After estimation, we determine the proper budget δ_0 according to Eq. 9.

Example 8 Consider again the sequence in Example 5. We use Adaptive-DP to repair this sequence with $w = 6$. The

Algorithm 2: Adaptive − DP(x, θ, w, α)

```
Data: data sequence $x$, error range $\theta$, window size $w$, threshold
      factor $\alpha$
Result: the maximum log probability of the optimal repair with
        adaptive budget
1  initialize $D(2, c_2, x'_1, x'_2)$ for each $x'_1, x'_2$;
2  $maxGain \leftarrow 0$;
3  $fitted \leftarrow false$;
4  for $c \leftarrow 0$ to $\infty$ do
5      for $i \leftarrow 3$ to $n$ do
6          Calculate $D(i, c, x'_{i-1}, x'_i)$ for $\forall x'_{i-1}, x'_i$;
7      $M^c(x) \leftarrow \max_{x'_{n-1}, x'_n} D(n, c, x'_{n-1}, x'_n)$;
8      $maxGain \leftarrow \max(maxGain, \frac{M^c(x)-M^{c-w}(x))}{w}$;
9      if $\neg fittecd \wedge [\frac{M^c(x)-M^{c-w}(x)}{w} < \alpha \cdot maxGain]$ then
10         Make the fitting and estimate $\rho$ and $\lambda$;
11         $fitted \leftarrow true$;
12         if $\{\delta | \frac{M^{\delta+w/2}(x)-M^{\delta-w/2}(x)}{w} < \alpha\rho\} \neq \emptyset$ then
13             $\delta_0 = \min\{\delta | \frac{M^{\delta+w/2}(x)-M^{\delta-w/2}(x)}{w} < \alpha\rho\}$;
14             return $M^{\delta_0}(x)$;
15     if $fitted \wedge [\frac{M^c(x)-M^{c-w}(x)}{w} < \alpha\rho]$ then
16         return $M^{c-w/2}(x)$;
```

operation of dynamic programming is basically the same as that of DP (see Example 4 and 5 of the conference version [48] for details). To select the proper budget, we first find that $\frac{M^6(x)-M^0(x)}{6} = 3.25$ is the maximum probability gain (see Fig. 8b). When $c = 16$, we find $\frac{M^c(x)-M^{c-w}(x)}{w} = 0.51 < 0.2 \times 3.25$ for the first time.

Therefore, we begin to fit with current data, i.e., $\frac{M^c(x)-M^{c-w}(x)}{w}$ for $c \in \{6, 7, \ldots, 16\}$. With $\rho = 3.33$, the threshold is $\alpha \cdot \rho = 0.67$. Since $\delta = 13$ is minimal for $\frac{M^{\delta+w/2}(x)-M^{\delta+w/2}(x)}{w}$, we finally regard $\delta_0 = 13$ as the proper budget.

Proposition 3 *Algorithm Adaptive-DP (Algorithm 2, the budget-adaptive version of DP) runs in $O(n\delta_0\theta_{\max}^3)$ time with $O(n\delta_0\theta_{\max}^2)$ space.*

Proof Algorithm DP runs in $O(n\delta_0\theta_{\max}^3)$ time with $O(n\delta_0\theta_{\max}^2)$ space. Additionally, looking for $M^c(x)$ for each c runs in $O(\theta_{\max}^2)$ time with $O(\delta_0)$ space and fitting runs in $O(\delta_0)$ time with $O(1)$ space. In conclusion, the overall complexity is still $O(n\delta_0\theta_{\max}^3)$ for time and $O(n\delta_0\theta_{\max}^2)$ for space. □

5.3.2 Adaptive-DPC

For the dual algorithm, we have an easier way to make it budget-adaptive. Removing the limitation of δ, i.e., $\delta \leftarrow \infty$, we run the original algorithm once. Then, by traversing $C(n, l'_n, x'_{n-1}, x'_n)$, we get the maximum log probability of arbitrary budget:

$$M^c(x) = \max\{l'_n | C(n, l'_n, x'_{n-1}, x'_n) = c\}$$

Thus, parameter ρ is estimated with all $M^c(x)$. Referring to Eq. 9, the adaptive budget can be easily got.

Proposition 4 *Algorithm Adaptive-DPC (the budget-adaptive version of DPC) runs in $O(n^2\theta_{\max}^3)$ time with $O(n^2\theta_{\max}^2)$ space.*

Proof Algorithm DPC runs in $O(n^2\theta_{\max}^3)$ time with $O(n^2\theta_{\max}^2)$ space. Additionally, calculating $M^c(x)$ costs $O(n\theta_{\max}^2)$ time with $O(1)$ space and fitting runs in $O(\delta_0)$ time with $O(1)$ space. In conclusion, the overall complexity is $O(n^2\theta_{\max}^3)$ for time and $O(n^2\theta_{\max}^2)$ for space. □

5.3.3 Adaptive-UG

For the universal greedy algorithms in Sect. 4.2, only the loop termination condition needs to be changed. After a repair is applied, we add the corresponding $G_i(x_i'')$ into the set $\mathcal{G}$. Similar to the above methods, we estimate ρ with the exponential-distributed $\mathcal{G}$ when the current $G_i(x_i'')$ is less than $\alpha \cdot \max\{\mathcal{G}\}$. Then, according to Eq. 9, $G_i(x_i'') < \alpha\rho$ will be the new loop termination condition.

Proposition 5 *Algorithm Adaptive-UG (the budget-adaptive version of UG) runs in $O((n+\delta_0)\theta_{\max}\log(n\,\theta_{\max}))$ time.*

Proof Algorithm UG (Algorithm 1) runs in $O((n+\delta_0)\theta_{\max}\log(n\theta_{\max}))$ time according to Proposition 1. Additionally, fitting runs in $O(\delta_0)$ time with $O(1)$ space. In conclusion, the overall time complexity is still $O((n+\delta_0)\theta_{\max}\log(n\theta_{\max}))$. □

6 Repair with moving windows

In this section, we discuss streaming repair with moving windows. There are two detailed algorithms introduced, which are based on scroll and sliding windows, respectively. Moreover, since the stream length is infinite, it is impossible to manually specify the budget, where the idea of budget adaptation in Sect. 5 helps.

6.1 Scroll algorithm

Scroll window is a widely used technology in many fields [31, 42]. Here, we propose the algorithms based on scroll windows, dividing the stream into windows and handling them independently.

Suppose the maximal latency is $d_{\max}$. The data points more than $d_{\max}$ earlier will be ignored. Practically, we set $d_{\max}$ based on the following criterion: First, very few latency is not covered by $d_{\max}$. Second, ignored data points are too late to influence the real-time decisions. Third, keeping too many data points leads to too much storage consumed.

The out-of-order stream x should be reordered by timestamps as x', i.e., $t_i' < t_j'$ for $\forall i < j$. Let x_i be a data point from input stream x; it is inserted into the ordered stream x' as x_l'.

With the idea of scroll window, we keep a fixed-size window whose size is W. If the timestamp of x_l' is later than the start of the window, x_l' will be added to the window. When the window is full, it will be solved by the adaptive algorithms in Sect. 5.3 with the current probability distribution and emptied. The probability distribution keeps unchanged inside each window. If x_l' is earlier than the start of the window, we traverse all possible repairs of x_l' and choose the one with the largest probability while other points are unchanged. Algorithm 3 shows the main procedure of scroll algorithms.

Algorithm 3: Scroll – Algorithm(x, θ, W)

```
Data: out-of-order data sequence $x$, error range $\theta$, window size $W$
Result: the repaired data sequence $x'$ ordered by timestamps
1  $st \leftarrow 1$;
2  for $i \leftarrow 1$ to $n$ do
3      insert $x_i$ into $x'$ as $x_l'$;
4      if $l < st$ then
5          $st \leftarrow st + 1$;
6          $x_l' \leftarrow \arg\max_{x_l''} G_l'(x_l'')$;
7      else if $i = st + W - 1$ then
8          Repair $x'_{st \ldots st+W-1}$ with adaptive algorithms;
9          $st \leftarrow st + W$;
10 return $x'$;
```

Example 9 **(Scroll Algorithm)** Consider the out-of-order sequence in Example 3 The probability distribution is shown in Fig. 3. We have $\theta_i = 2$ and $W = 6$. When the first 6 data points come, the window is full. We have $x'[1\ldots6] = \{11, 12, 15, 14, 15, 17\}$ with $t'[1\ldots6] = \{1, 2, 3, 4, 5, 7\}$. After repairing, we have $x'[1\ldots6] = \{11, 12, 13, 14, 15, 17\}$.

For x_7, it is inserted as x_6'. Traverse $x_6' \in \{13, 14, 15, 16, 17\}$, we find that the probability reaches maximum when $x_6' = 16$. Finally, the returned sequence is $x' = \{11, 12, 13, 14, 15, 16, 17\}$ with $t' = \{1, 2, 3, 4, 5, 6, 7\}$.

6.2 Sliding algorithm

Similarly, sliding window is also a widely used strategy [10, 18]. With the sliding idea, we design a greedy algorithm. This algorithm is incremental, i.e., point in, point out, which has two main steps.

The first step is reordering just like that in scroll algorithms. The second step is repairing. We use a sliding window containing the last W data points in x'. When a data point x_p' leaves the window, we traverse and calculate the probability gain $G_p'(x_p'')$ for all of its possible repairs x_p''. The one with the largest gain is chosen as the final repair.

To avoid over-repairing, referring to the idea of Adaptive-UG, the adaptive budget in Problem 2 is transformed to a probability gain threshold. The repair whose probability gain is below the threshold will be pruned. As shown in Sect. 5.2, we add the probability gain of each applied repair into the set $\mathcal{G}$, fit it and dynamically set the threshold as $\alpha\rho$. To reduce the complexity, we only keep the most recent applied gains in $\mathcal{G}$.

Algorithm 4 shows how the incremental greedy algorithm based on sliding window works. Once a data point x'_p is repaired, its value will never be changed again. There are at most W points unrepaired in x' at the same time. Moreover, for each coming data point, the time and space cost of IG are both unrelated to data size.

Algorithm 4: IG(x, θ, α, W)

```
Data: out-of-order data sequence $x$, error range $\theta$, threshold
      factor $\alpha$, window size $W$
Result: the repaired data sequence $x'$ ordered by timestamps
1  $x' \leftarrow \emptyset$;
2  $\mathcal{G} \leftarrow \emptyset$;
3  $threshold \leftarrow 0$;
4  for $i \leftarrow 1$ to $n$ do
5      insert $x_i$ into $x'$ as $x'_l$;
6      $p \leftarrow \min(l, i - W)$;
7      if $\max_{x''_p} G'_p(x''_p) > threshold$ then
8          $x'_p \leftarrow \arg\max_{x''_p} G'_p(x''_p)$;
9          Update $\mathcal{G}$ with $\{\max_{x''_p} G'_p(x''_p)\}$;
10         Make the fitting with $\mathcal{G}$ and estimate $\rho$ and $\lambda$;
11         $threshold \leftarrow \alpha\rho$;
12 return $x'$;
```

Example 10 **(Incremental Greedy)** Consider again the sequence in Example 9 and the window size $W = 2$. Suppose the threshold is statically set to 0.5 due to the converged fitting. The first four data points are reordered as $x' = \{11, 12, 15, 14\}$ and $t' = \{1, 2, 3, 4\}$. At the same time, we try to repair x'_1 and x'_2 but no probability increase can be made.

After that, x_5 is inserted as x'_5. We have $x' = \{11, 12, 15, 14, 15\}$ and $t' = \{1, 2, 3, 4, 5\}$. Thus, x'_3 should be repaired because $3 = \min(5, 5-2)$. Traverse all possible values, x'_3 is repaired to 13 with the probability gain 1.4. Since $1.4 > 0.5$, this repair will be accepted.

Similarly, we can reorder and repair the entire sequence. At last, the repaired sequence is $x' = \{11, 12, 13, 14, 14, 15, 17\}$ with $t' = \{1, 2, 3, 4, 5, 6, 7\}$.

Proposition 6 *For each coming data point, Algorithm IG runs in $O(\log d_{\max} + \theta_{\max} + |\mathcal{G}|)$ time with $O(|\mathcal{G}| + \max\{d_{\max}, W\})$ space.*

Proof For each data point, the time cost can be divided into three parts. First, inserting into the correct place needs $O(\log d_{\max})$ time with dichotomy. Second, finding the repair with maximal probability gain needs $O(\theta_{\max})$ time by traversing all possible repairs. Third, fitting needs $O(|\mathcal{G}|)$ time. Therefore, the total time cost is $O(\log d_{\max} + \theta_{\max} + |\mathcal{G}|)$.

As for space complexity, the space of recent data points is $O(\max\{d_{\max}, W\})$. Additionally, with the set $\mathcal{G}$, the total space is $O(|\mathcal{G}| + \max\{d_{\max}, W\})$. □

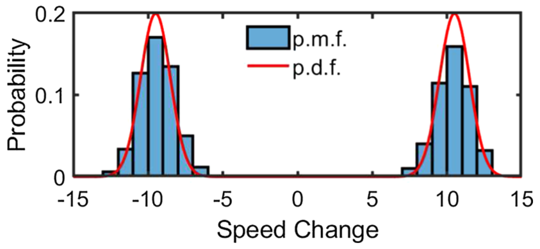

Fig. 10 Distribution of speed changes over SYNTHETIC dataset

7 Experimental evaluation

7.1 Experiment setup

7.1.1 Algorithm

As for repairing as a whole, we experimentally compare our proposed methods, DP and DPC in Sect. 4.1, UG in Sect. 4.2, DPL, QP and SG in conference version [48] with SCREEN the state-of-the-art approach [35]. We omit to report the other methods, such as the smoothing-based EWMA [16], or the constraint-based [17], owing to the clearly worse results (which are also observed in [35]).

Moreover, we compare our budget-adaptive methods in Sect. 5.3, including Adaptive-DP, Adaptive-DPC and Adaptive-UG. Because the output of SCREEN is unrelated to the budget, we also use SCREEN as a baseline.

As for streaming repairing, the scroll algorithms Scroll-DP, Scroll-DPC, Scroll-UG in Sect. 6.1 and the incremental algorithm IG in Sect. 6.2 will be compared. Similarly, SCREEN is the baseline since it supports stream data. Besides, REMIAN [27] repairing with the correlations between various dimensions is also used as baseline.

7.1.2 Dataset

In this section, the experiments run on five real datasets and two synthetic datasets, whose speed change distributions are shown in Fig. 10 and 11.

– **STOCK**: This dataset records the daily prices of a stock from 1984-09 to 2010-02, with 12826 data points in total.

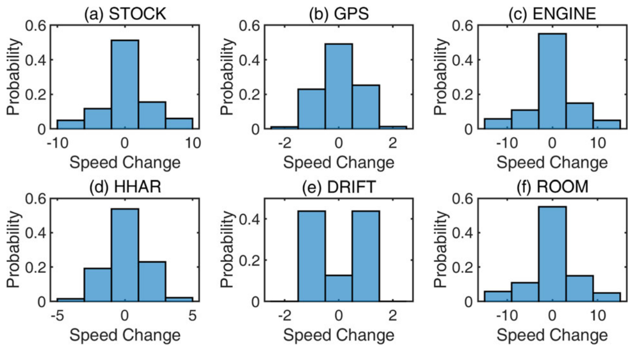

Fig. 11 Distributions of speed changes over various datasets

Since the data is originally clean, errors are injected following the same line of evaluating the repair effectiveness [6]. Each data point is picked as an error randomly and independently. Then, the error points are added by a random number following the uniform distribution within $[-\theta, \theta]$, where θ is the given error range.

- **GPS**: This dataset is collected by a person carrying a smartphone and walking around campus. Instead of synthetically injecting errors, *true errors* are naturally embedded in the dataset. Since we know exactly the path of walking, a number of 150 dirty points are manually identified (among total 2,358 clear points in the trajectory). True locations of dirty points are also manually labeled as ground truth.
- **ENGINE**: This dataset collects four sequences of a crane, produced by a heavy industry company, which monitors the working status of the device. The meaning of variables and their domains are presented in Table 4. The total number of data points in each sequence is 464. Since no error and truth are known in advance, this dataset is only used for regression. We evaluate the prediction of switching-count by sensor readings with/without repairing. The switching-count observation serves as the ground truth of prediction.
- **SYNTHETIC**: This dataset is generated by ourselves. First, we generate the ground truth with speed changes sampled from the probabilistic model[5] shown in Fig. 10. Then, we inject errors into ground truth, following the same line over STOCK, with error range $\theta = 5$, error number 100, and data size 500. To perform the prediction experiments, we generate other two determinant sequences plus one dependent sequence.
- **HHAR**: This dataset records the acceleration collected from smartphones [36]. There are 70000 data points in this dataset. Since there is a small gap in the creation time and arrival time, this dataset is *naturally out of order*. Errors are randomly injected following the same line over STOCK, with error range $\theta = 10$ and error number 7000.
- **DRIFT**: This dataset is generated with Massive Online Analysis [4], which contains three parts. The speed changes in the first and third parts are sampled from $\{-1, 1\}$ while those in the second part are sampled from $\{-1, 0, 1\}$. The data size of each part is 5000 and 15000 totally. There is a concept shift between the first and second parts, as well as a slow drift with width 1000 between the second and third parts. The speed change varying with time is shown in Fig. 33a. Errors are randomly injected following the same line over STOCK, with error range $\theta = 5$ and error number 750.
- **ROOM**: This dataset records temperature, relative humidity and CO2 in a room [8]. It has four sequences with data size 8143. We randomly inject errors into the first sequence temperature, following the same line over STOCK, with error range $\theta = 5$ and error number 1000.

7.1.3 Environment

We implement all algorithms with Java 8 and run them on a Java HotSpot(TM) 64-Bit Server VM. The physical machine is a PC with an Intel(R) Core(TM) i7-9700 CPU and 16GB RAM.

7.2 Evaluation on repairing as a whole

In our first experiment, we suppose the entire sequence of STOCK dataset[6] is in an independent window and compare our repairing algorithms with the baseline. Let x_{truth} be the ground truth of clean sequence, and x_{repair} be the repaired sequence. The repair accuracy is measured by root-mean-square error (RMS) [21], evaluating how close the repaired sequence x_{repair} is to the ground truth x_{truth}. The lower the RMS error is, the closer (more accurate) the repair is to the ground truth. Besides the RMS performance on repairing accuracy, we also report the corresponding time cost.

7.2.1 Error range

For the original clean STOCK dataset, we have error range $\theta = 0$ (i.e., no need to repair). To evaluate the repair performance, we manually inject errors into the dataset with error range $\theta > 0$. Figure 12 presents the results by varying the ranges of injected errors from $\theta = 1$ to 10. The error number is set to 600 and the budget is set to the expectation of $\Delta(x, x_{truth})$, i.e., $\frac{1}{2} \times \#Error \times \theta = 300\theta$. We set $d = 1000$ for DPL and $\varepsilon = 2$ for DPC.

[5] Mixed Gaussian distribution whose density function is $\mathcal{N}(-10, 1) + \mathcal{N}(10, 1)$.

[6] To save experiment time, we only use part of the dataset in some experiments.

Table 4 ENGINE variables

Variable	Description	Domain
DT0	Current of a proportioner called DT0	[200, 800]
Engine-speed	Rotate speed of the engine	[800, 2000]
Pump-volume	Swept volume of the pump	[0, 100]
Switching-count	Times the crane pumping per minute	[3, 27]

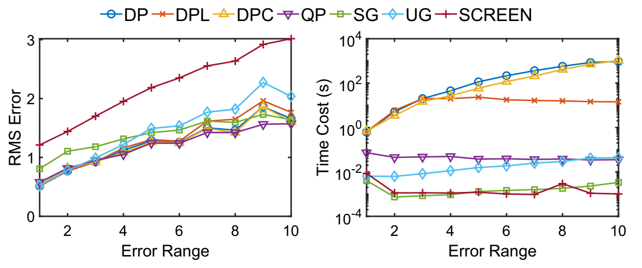

Fig. 12 Varying error range θ, over STOCK with error number 600 and data size 1282

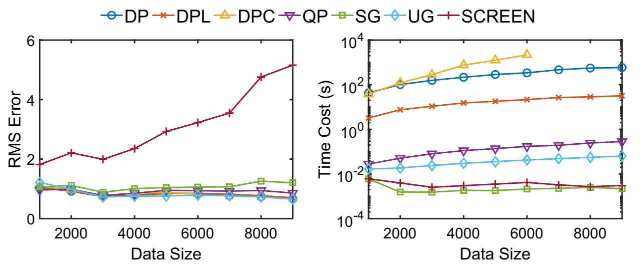

Fig. 14 Scalability, over STOCK with error range $\theta = 5$ and error number 400

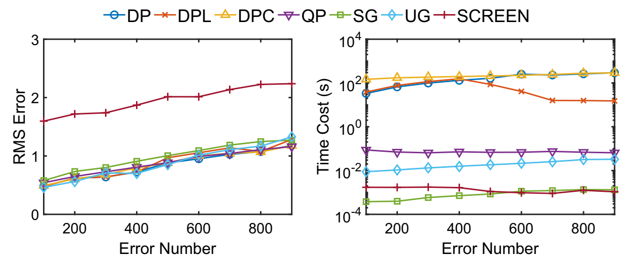

Fig. 13 Varying error numbers, over STOCK with error range $\theta = 5$ and data size 2564

First, as shown in Fig. 12, it is not surprising that the larger the error range is, the higher the RMS distance will be between the truth and repair results.

Second, although with the strong simplification in Eq. 2, all of our repairing algorithms show much lower RMS than baseline SCREEN. Among them, SG and UG show a bit higher RMS than other algorithms, but their time costs are lower.

7.2.2 Error number

Fig. 13 reports the results on various number of errors that are injected into the data. The error range θ is set to 5. Similarly, we have $\delta = \frac{5}{2} \cdot \#Error$, $d = 1000$ and $\varepsilon = 2$.

Generally, the RMS error increases with the increasing number of errors. The existing method SCREEN has a much worse RMS measure than all of the other algorithms. Through a more detailed comparison, DP, DPL and DPC have slightly lower RMS than QP, SG and UG, with the cost of greater time overhead.

Moreover, the time cost of DP is lower than that of DPC with small error number in Fig. 13. The reason is that with a small error number, the corresponding repair budget δ needed is small as well. The DP algorithm with $O(n\theta_{\max}^3\delta)$ complexity runs faster as well. For the same reason, in Fig. 13, a larger number of errors lead to larger δ, and thus DP shows higher time cost (closer to that of DPC).

It's interesting that DPL runs faster with a large error number. When $\delta \le d$, DPL degenerates to be the same as DP. After that, we have $H = \delta/d$ and the search space of candidates is reduced to θ/H, which means that a larger number of errors lead to lower time cost. Therefore, in Fig. 13, the time cost of DPL rises at first and falls then.

7.2.3 Data size

Figure 14 presents the results over various data sizes. The budget δ is fixed to 1000 with $\theta = 5$ and $\#Error = 400$. We set $d = 500$ and $\varepsilon = 2$.

In Fig. 14, the RMS of SCREEN has risen sharply. On the contrary, the RMS performance of our algorithms is stable with different data sizes, which indicates the potential for dealing with big data issues.

However, the results of DPC with data size above 7000 are not marked in Fig. 14 because the programs crash halfway out of memory. The space complexity of DPC is $O(n^2\theta_{\max}^2)$, which is very sensitive to data size. Thus, DPC is not suitable to repair in a large window.

7.2.4 Bin width

As shown in Sect. 3.1, we use a binning method to construct the probability distribution of speed change. Our repairing algorithms work well when each bin has a sufficiently high probability. Figure 15a gives an example of the probability

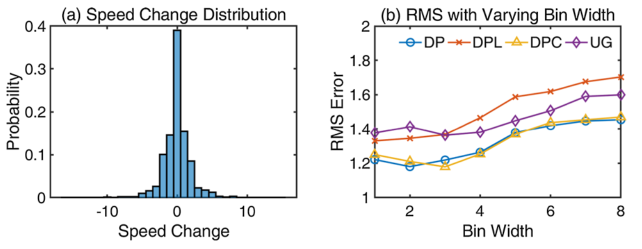

Fig. 15 Varying bin width over STOCK with error range $\theta = 5$, error number 600 and data size 1282

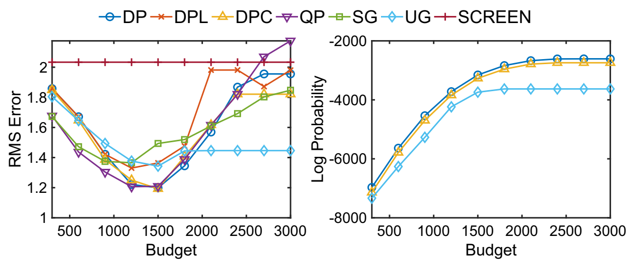

Fig. 16 Varying repair cost budget δ from 300 to 3000, over STOCK with error range $\theta = 5$, error number 600 and data size 1282

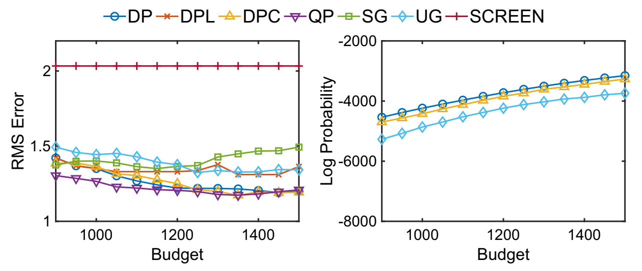

Fig. 17 Detailed results with δ in the range of 900 to 1500 in Fig. 16

distribution over STOCK with bin width 1. Each bin has a sufficiently high probability.

Therefore, bin width is an important factor that influences the construction and thus the final repair. Figure 15b shows the RMS with varying bin width over STOCK. QP, SG and SCREEN are not plotted because they are unrelated to probability distribution. The budget is fixed to $\delta = 1200$. When the bin width is smaller than 4, the robustness of RMS makes it not hard to specify the width. On the contrary, if the bin width is very large, nearly all probability falls into the same bin, leading to no repair and high RMS.

7.3 Evaluation on budget-adaptive algorithm

7.3.1 STOCK with injected errors

Figure 16 reports the results of STOCK dataset by varying the repair cost budget δ. If the budget is set too small, the dirty points cannot be fully repaired, with a higher RMS measure. The corresponding log probability of repair results is low as well. On the other hand, if the budget δ is set too large, the sequence might be over-repaired. The RMS measure is high as well. Thus, the guideline for setting the proper δ in Sect. 5.1 is verified again. Note that by further increasing the budget, the log probability could not increase further in Fig. 16. The reason is that the repaired sequence reaches the allowed repair range θ. The corresponding RMS measure does not change either.

It is worth noting that a range of δ could be considered. We illustrate a very large range of δ in Fig. 16 in order to verify the guideline of choosing a proper δ. Once a proper range of δ is identified, e.g., from 900 to 1500 where the log probability no longer greatly increases, the repair results are stable. To demonstrate the robustness, Fig. 17 shows the results with δ in the range of 900 to 1500. As shown, the result appears to be less sensitive in the chosen range of δ under the aforesaid guideline. It is helpful for adaptive algorithms to handle noises.

According to the aforesaid guideline, we propose several budget-adaptive algorithms in Sect. 5.3. Table 5 shows the performance. Referring to Fig. 16, these budget-adaptive algorithms succeed in selecting proper δ. Especially, by enumerating the budget δ over STOCK, the lowest RMS of DP and DPC are both 1.19, while Adaptive-DP and Adaptive-DPC achieve the RMS of 1.22 and 1.20, respectively. The results of adaptive algorithms are very close to the best ones, which effectively reduces the burden of manual budget-decision.

Moreover, we randomly insert 20% of errors with error range $\theta = 5$ into the data. Figure 18 compares the adaptive and optimal RMS and budget. The optimal results are received by traversing all possible budgets. With different data sizes, the adaptive algorithms can always find the proper budget and achieve low RMS close to the optimal one.

7.3.2 GPS with naturally embedded errors

In GPS dataset, the results are similar. Figure 19 reports the RMS measure and log probability of results, by varying the repair cost budget δ. Similar to Fig. 16, when the budget δ is small, dirty data might not be sufficiently repaired and thus the log probability of results is low. By granting more budget (larger δ), while the data could be over-repaired (higher RMS error), the log probability does not greatly increase further. Together with the results in Table 5, the effectiveness of the budget-determination guideline and budget-adaptive algorithms is verified again.

7.3.3 ENGINE in real application

Repairing on the real ENGINE dataset can be regarded as a real *application*. This dataset includes four attributes, DT0,

Table 5 The performance of adaptive algorithms over STOCK, GPS, ENGINE and SYNTHETIC

	Stock		GPS		Engine			Synthetic		
	RMS	Time cost	RMS	Time cost	RMS	R^2	Time cost	RMS	R^2	Time cost
Adaptive-DP	1.22	127*s*	0.447	17.0 min	2.2509	0.79806	20.6 min	0.730	1–3.7e−9	7.40*s*
Adaptive-DPC	1.20	77.7*s*	0.493	120 *min*	2.2509	0.79806	6.03 min	0.831	1–4.7e−9	6.28*s*
Adaptive-UG	1.38	53.1 ms	0.395	233 ms	2.2500	0.79822	37.1*ms*	0.828	1–4.7e−9	9.95 ms
SCREEN	2.03	5.96 ms	0.496	9.64 ms	2.2559	0.79717	2.36ms	1.62	1–1.8e−8	2.36 ms
RAW	–	–	–	–	2.2513	0.79800	396μs	1.48	1–1.5e−8	304 µs

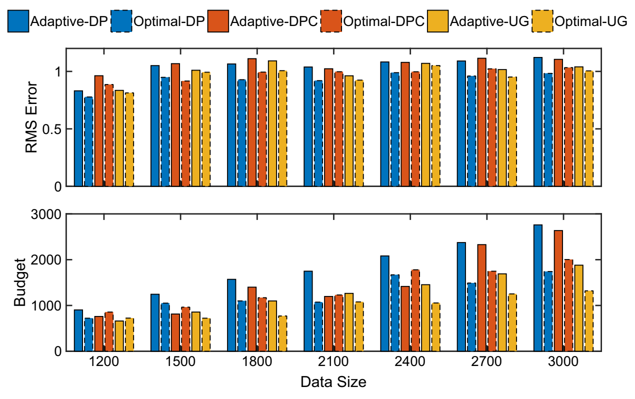

Fig. 18 The comparison between adaptive and optimal RMS/budget with varying data size

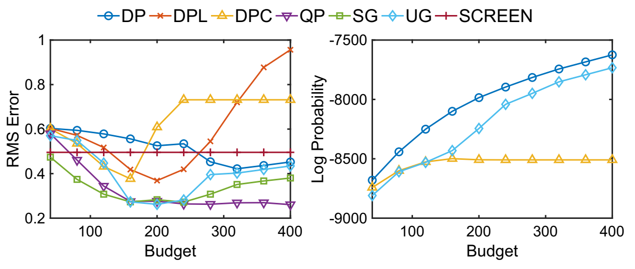

Fig. 19 Varying repair cost budget δ from 40 to 400 over GPS

engine-speed, pump-volume and switching-count. Owing to the sensor issues, the readings of engine-speed are often inaccurate and thus need repairing. Since switching-count is often missing in practice, the application is to predict switching-count with the multiple linear regression model.

$$\text{switching-count} = \gamma_1 * \text{pump-volume} + \gamma_2 * \text{DT0} + \gamma_3 * \text{engine-speed} + \gamma_4$$

where γ_1 to γ_4 are parameters to estimate.

To perform the prediction, we use the linear least square method. To evaluate the application accuracy, two measures are employed, RMS reporting the closeness of the predicted values to the observed switching-count values (that are not missing) and R^2 the coefficient of determination [13]. A lower RMS error or a higher R^2 measure denotes better prediction accuracy.

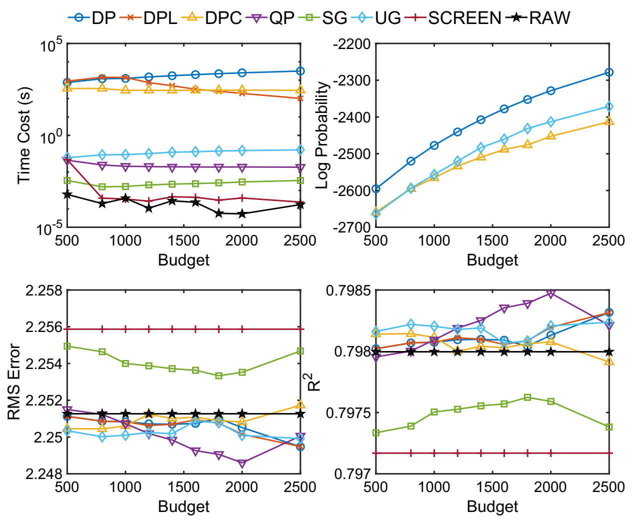

Fig. 20 Varying repair budget δ over ENGINE

Figure 20 illustrates the results by varying the budget δ. As shown, by a proper set of budget δ, the RMS and R^2 of most prediction application improve, compared to RAW without repairing.

As shown in Fig. 20, the existing SCREEN repair leads to higher RMS error in prediction. It is not surprising because the similar results of SCREEN are shown in Fig. 16 over STOCK and Fig. 19 over GPS. Similarly, the other prediction measure R^2 of SCREEN is lower, which verifies again the results over other datasets.

Besides, Table 5 shows the performance of adaptive algorithms. Adaptive-DP, Adaptive-DPC and Adaptive-UG all achieve better accuracy than the baselines, indicating that the adaptive strategy can really select the proper budget.

7.3.4 SYNTHETIC on regression

Similar to those over ENGINE dataset, we evaluate the performance of repairing on multiple linear regression over SYNTHETIC.

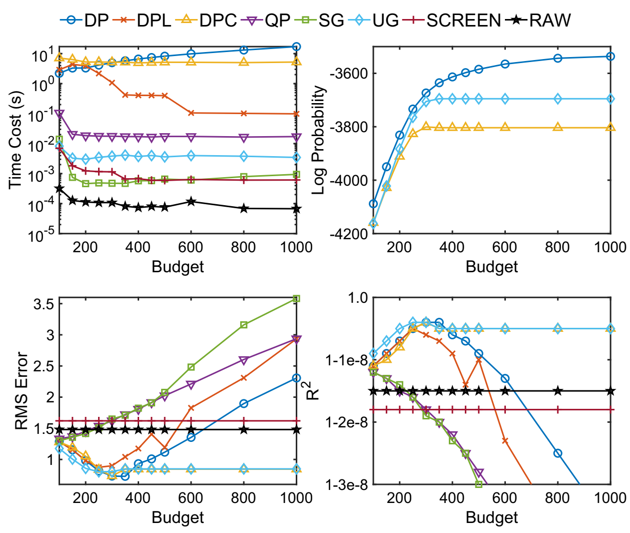

Fig. 21 Varying repair budget δ over SYNTHETIC

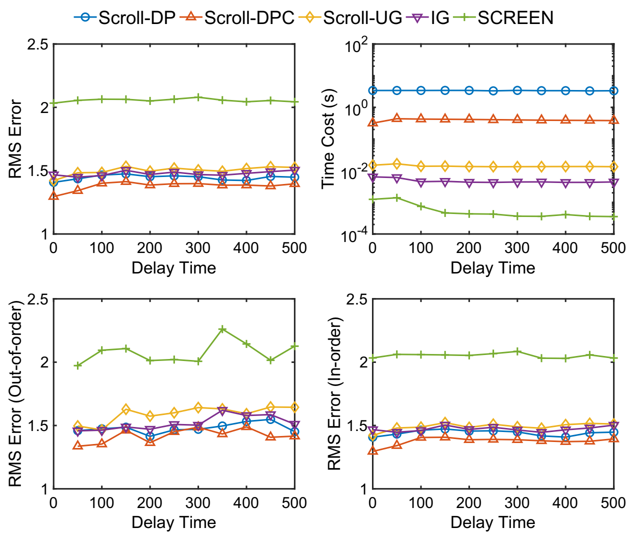

Fig. 22 Varying delay time from 0 to 500 with delay rate 10% over STOCK

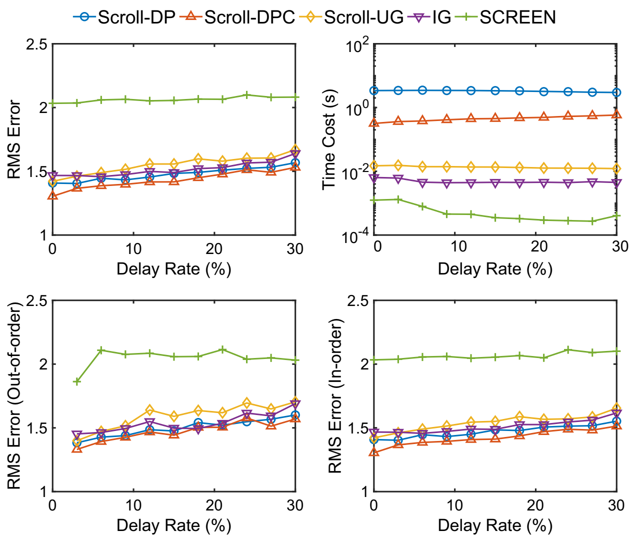

Fig. 23 Varying delay rate from 0 to 30% with delay time 200 over STOCK

Figure 21 presents the prediction results. SCREEN has the highest RMS error and other repairing algorithms achieve much better performance. Moreover, we discover that the RMS errors of QP and SG are high and rise with the increasing budget. The reason is that the speed changes shown in Fig. 10 are not Gaussian-distributed, which destroys the foundation of QP and SG and leads to poor results. On the other side, the low RMS errors of DP, DPL, DPC and UG indicate their universality in probability distribution.

As for the adaptive algorithms, Table 5 shows the similar results with the above datasets. Especially, the lowest RMS in Fig. 21 of DP is 0.729, which is very close to the RMS of Adaptive-DP. Clearly, the adaptive methods perform well in SYNTHETIC dataset.

7.4 Evaluation on streaming repairing

Next, we evaluate the performance of streaming repairing. To test the solutions for out-of-order arrival, we use the dataset HHAR which is naturally out-of-order. Moreover, we also make the originally in-order datasets STOCK and GPS out of order, following the method in [35]. First, we randomly pick up part of the data points (the percentage is called *delay rate*) and then add a uniformly distributed random number (the maximum is called *delay time*) to the original timestamps as arrival timestamps. The arrival timestamps of other data points are still equal to the original ones. Finally, the sequence is reordered according to arrival timestamps.

7.4.1 Delay

We evaluate the performance of our streaming cleaning algorithms over real datasets STOCK and GPS with varying *delay rate* and *delay time*. Four evaluation criteria are used in the evaluation. Besides the overall RMS and time cost, we also report the RMS for in-order and out-of-order points, respectively.

For STOCK dataset (error range $\theta = 5$, error number 600 and data size 1282), the window size is $W = 30$. Importantly, when the delay rate or delay time is equal to 0, the sequence is in order. There is no large difference between the results of in-order and out-of-order sequences, showing that our streaming algorithms can effectively repair both of them.

In Fig. 22, the RMS of out-of-order points is a bit higher than in-order points. For scroll algorithms, if out-of-order points are not in the scroll window, it is repaired with the different methods from in-order points. Thus, with increasing delay time, more out-of-order data points are out of

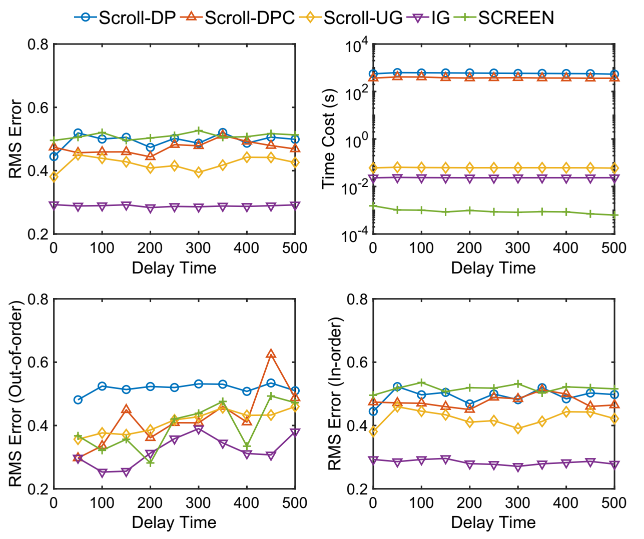

Fig. 24 Varying delay time from 0 to 500 with delay rate 10% over GPS

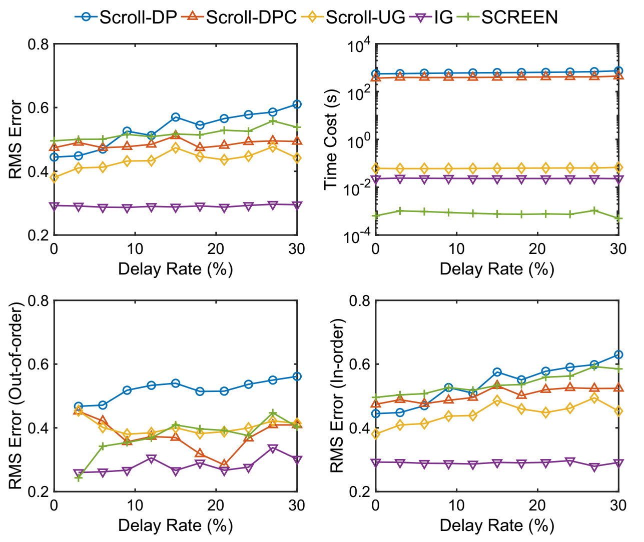

Fig. 25 Varying delay rate from 0 to 30% with delay time 200 over GPS

window leading to increasing RMS of out-of-order points. However, because the delay rate is only 10% and RMS of in-order points is stable, the overall RMS is also stable. In Fig. 23, RMS of both the out-of-order and in-order data points increases slightly with delay rate, leading to increasing overall RMS.

Among all algorithms, the baseline SCREEN has the worst RMS as repairing as a whole. In detail, Scroll-DP achieves the lowest RMS with the largest time cost. Scroll-UG and IG run much faster at the cost of only a little bit of accuracy.

The results over GPS are shown in Figs. 24 and 25. The window size is set as $W = 300$. Similarly, a high delay time leads to high RMS of out-of-order points and high delay rate rises the RMS of both points. But in this dataset, IG achieves

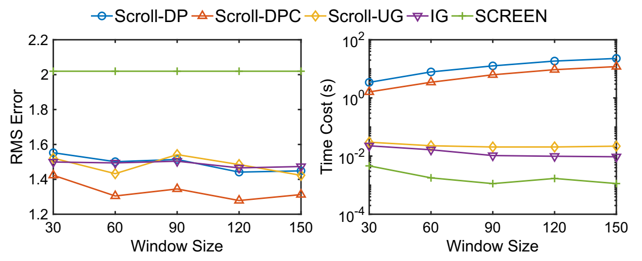

Fig. 26 Varying window size from 30 to 150 over STOCK

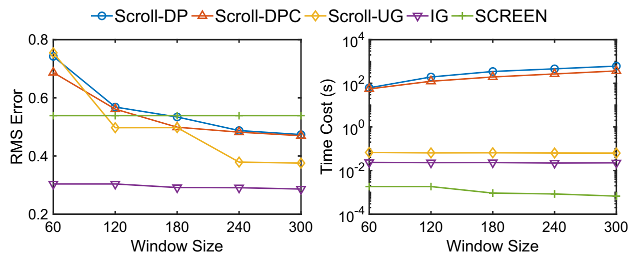

Fig. 27 Varying window size from 60 to 300 over GPS

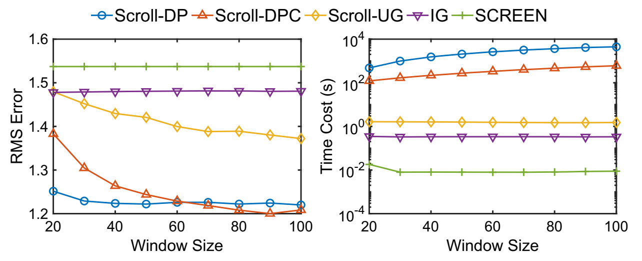

Fig. 28 Varying window size from 20 to 100 over HHAR

the lowest RMS and its performance is stable with different delay time and rate, showing the superiority of our streaming algorithms.

7.4.2 Window size

Window size W is a key parameter for our streaming algorithms. For dataset STOCK and GPS, we set the delay time as 200 and the delay rate as 10%. Figure 26 and Fig. 27 show the performance with varying window size W.

For scroll algorithms, the connection between different windows is broken. Thus, a larger window leads to more effective use of local information, which decreases the RMS at the cost of time. For sliding algorithm IG, increasing the window size can also decrease RMS a bit.

As for the naturally out-of-order dataset HHAR, Fig. 28 shows the results. SCREEN achieves the highest RMS showing the effectiveness of our algorithms. Moreover, the performance of IG is not sensitive to the window size because the delay in HHAR is short and rare; thus, a small window is sufficient.

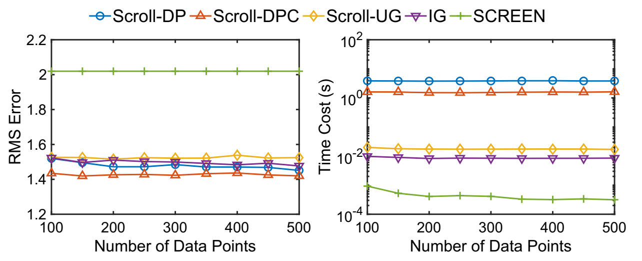

Fig. 29 Varying the number of data points for probability distribution construction over STOCK

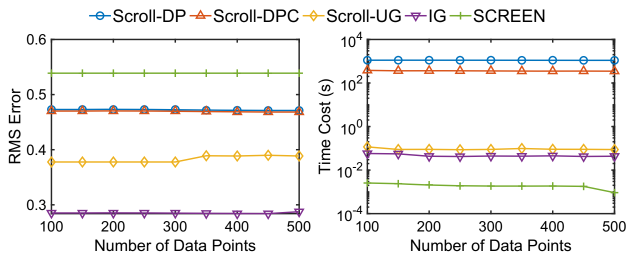

Fig. 30 Varying the number of data points for probability distribution construction over GPS

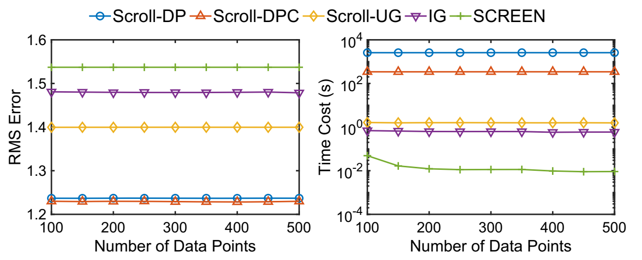

Fig. 31 Varying the number of data points for probability distribution construction over HHAR

All in all, with proper window size, all of our algorithms achieve lower RMS than the baseline SCREEN. It illustrates that streaming data can be effectively repaired by our algorithms.

7.4.3 Probability distribution construction

As shown in Sect. 3.2, we first construct the probability distribution with some data points and gradually update it when new data point arrives. Over the same datasets as Sect. 7.4.2, we evaluate the influence of the number of data points for probability distribution construction.

In dataset STOCK, we set window size $W = 30$. The performance of our streaming algorithms is robust with varying number of data points as shown in Fig. 29. Clearly, the robustness greatly simplifies the probability construction. Meanwhile, similar results are shown in Fig. 30 and Fig. 31 for GPS ($W = 300$) and HHAR ($W = 60$), respectively.

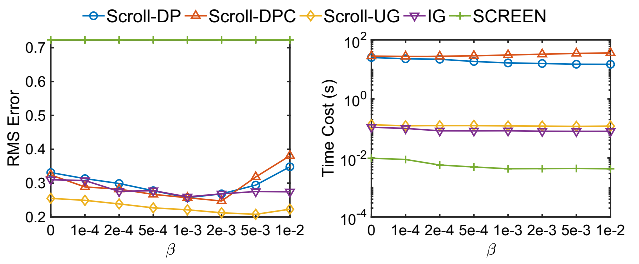

Fig. 32 Varying decay factor β from 0 to 0.01 over DRIFT

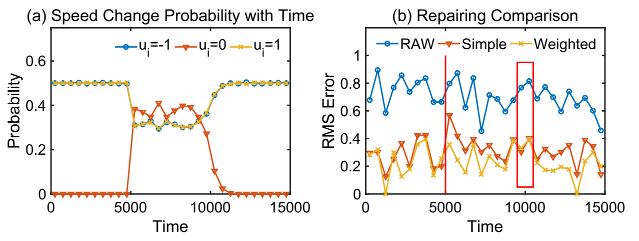

Fig. 33 Example of concept shift and drift with Scroll-DPC and $\beta = 0.001$

7.4.4 Concept shift and drift

Concept shift and drift are common in streaming data and harmful to our algorithms. As introduced in Sect. 3.3, our solution is weighted probability construction. The comparison is between simple ($\beta = 0$) and weighted probability construction over dataset DRIFT.

Figure 32 shows the RMS and time cost of streaming algorithms with varying β. The time cost is little related to β. As for accuracy, compared to simple construction equally weights all data points, weighted construction gives more weight to closer data points, i.e., it focuses more on new distribution after concept shift or slow drift. Thus, it suffers less and achieves lower RMS. Meanwhile, too large β limits the data points for probability construction, which is also harmful to repairing accuracy. Besides, our algorithms outperform SCREEN a lot.

Figure 33 gives a detailed comparison between simple and weighted construction, using Scroll-DPC. As shown in Fig. 33a, there is a concept shift at time 5000 and a slow drift around time 10000. In Fig. 33b, we plot the RMS every 500 data points. RAW, Simple and Weighted correspond to no repair, repair with simple probability construction with equal weight in Sect. 3.2 and repair with weighted probability construction in Sect. 3.3, respectively. At time 0-5000, the RMS is homogeneous between two constructions. After the concept shift at time 5000, the RMS of weighted construction is always lower than simple construction, because weighted construction turns to the new distribution while the simple one is still trapped by the old one. Similar results are also shown after the slow drift around time 10000.

Table 6 The performance of streaming algorithms compared with REMIAN over ROOM

	ROOM	
	RMS error	Time cost
REMIAN	1.082	31.4*ms*
Scroll-DP	0.544	12.8*s*
Scroll-DPC	0.558	15.8*s*
Scroll-UG	0.344	71.4*ms*
IG	0.400	46.4*ms*

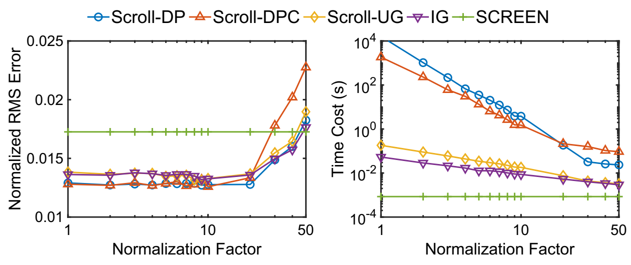

Fig. 34 Varying normalization factor over STOCK

7.4.5 Comparison with multi-dimension repairing

As shown in Problem 2, our algorithms maximize the probability based on speed changes. It captures the inner feature of a single-dimension time sequence. On the contrary, some algorithms capture the correlations of the multidimensional stream for repairing. For example, REMIAN [27] builds multivariate linear regression models to impute the missing values and repair the anomalous values in the streaming scenario.

Over multi-dimension dataset ROOM, REMIAN repairs temperature using other dimensions, while our algorithms repair temperature alone. Table 6 shows the result of the comparison. Because the correlations between various dimensions in dataset ROOM are not strong and REMIAN ignores the inner feature of a single dimension, the RMS of our algorithms is much lower than REMIAN. Besides, the time costs of algorithms based on greedy such as Scroll-UG and IG are comparable to REMIAN.

7.4.6 Normalization

Figure 34 shows the results of repairing with normalization in the streaming scenario. It is not surprising that the time cost is greatly reduced with a larger normalization factor H, i.e., fewer candidates considered. Meanwhile, compared to no normalization ($H = 1$), there is little accuracy loss with a moderately large factor, e.g., $H \le 10$.

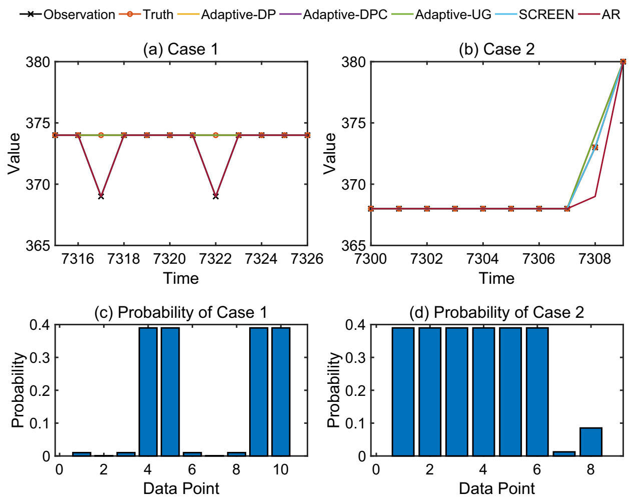

Fig. 35 Case study on STOCK

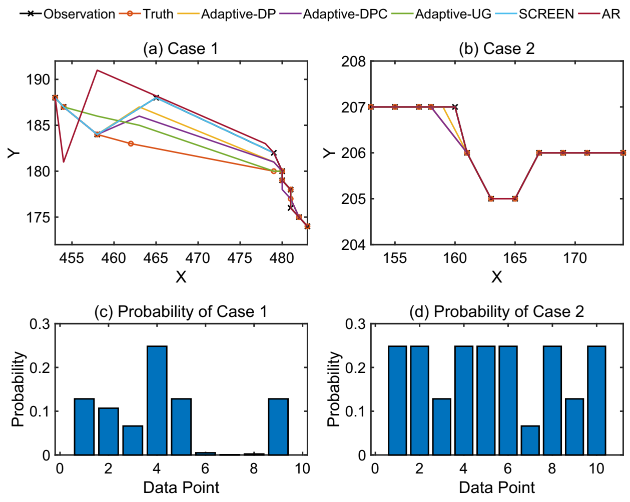

Fig. 36 Case study on GPS trajectory

7.5 Case study

7.5.1 Repair case over STOCK and GPS

To show the effectiveness of our repairing methods, we make a case study on repairing stock data. We compare our proposal with an outlier detection method, autoregressive model AR [20]. With the AR model learned on all data points, it first detects the observed values whose difference from the predicted value is larger than the threshold and identifies them as the outlier. Then, they are amended to predicted values.

Figure 35a shows an example of inserted errors, while Fig. 35c shows the probability of the observed speed change before and after each point. Since the inserted errors are rare in sequence, the probability of data points 1–3 and 6–8 is quite low and our adaptive algorithms successfully repair them. Besides, Fig 35b shows a case of error-in-the-system.

On the back of positive news, stock price rises rapidly. Since such a rise and fall of stock price may occur multiple times, the probability shown in Fig 35d is higher than Case 1. Thus, the sequence is not (or slighted) modified.

Similarly, we also make a similar study on repairing GPS trajectory. Figure 36a shows a case of error-in-the-measuring-instrument, while Fig. 36c shows the probability. Due to the sensor failures, the observed trajectory (black line, but covered by the blue line) is far from the ground truth. Therefore, based on the distribution of speed changes, the corresponding probabilities are very low, such as data points 6–8 in Fig. 36c. Figure 36b shows a case of error-in-the-system. Because there is an obstacle in front of the road, every walker has to change the path and bypass it. Different from the speed changes in error-in-the-measuring-instrument that rarely occur, every walker behaves similarly in this place and thus leads to a relatively higher probability, as illustrated in data points 3–8 in Fig. 36d.

For Case 1 shown in Fig. 36a, our methods repair the trajectory and make it closer to the ground truth. Unfortunately, AR fails to identify the dirty values but modifies the clean values by mistake, since it cannot capture the information on speed changes. For Case 2 shown in Fig. 36b, all algorithms do not (or slightly) repair the sequence. In summary, while both methods lean to leave the error-in-the-system unchanged in (b), our proposals repair more accurately the error-in-the-system in (a).

Although, over STOCK and GPS, we observe that the error-in-the-system is more frequent with a higher probability in the distribution, while the error-in-the-measuring-instrument occurs randomly leading to a lower probability, it might not be the case in some other practices. In that case, we have no idea which likelihood is higher, but we can assume that the distributions are different. For instance, we may assume that the distributions are Gaussian mixtures in both cases, and each one is governed by different parameters. In that context, extra knowledge could be employed to further distinguish which one is error-in-the-system. Since we do not have an assumption of Gaussian distribution in this study, we leave this interesting direction for future work.

7.5.2 Confusion matrix over GPS

From the above repair cases, we verify the conclusion in Sect. 1. SCREEN is good at detecting large spike errors rather than small errors. Meanwhile, over-repairing of smoothing methods is indicated in Sect. 1 as well. A statistical result in Table 7 also confirms them, where TP/FN is for errors with/without repairing and FP/TN is for correct data with/without repairing.

With a very high FP rate, smoothing method SMA modifies almost all data points. On the contrary, SCREEN has the lowest FP rate among all algorithms, because it only repairs the large spikes which are more likely to be errors. Our algorithms can detect both small and large errors, and avoid over-repairing with adaptive budget determination. Therefore, F-score shows the much better overall performance of our algorithms.

8 Related work

8.1 Constraint-based repairing

Constraint is important in data repairing. Based on various constraints, it is able to find and repair the errors. For event sequences, the constraints can be extracted from the flow chart [32, 40, 41]. Meanwhile, we can repair the rational data based on functional dependencies [26].

For time series, sequential dependencies proposed in [17] consider the range of value changes between two consecutive data points, while the distances on timestamps between data points are not involved. Even further, SCREEN [35] considers more universal speed constraints. In this sense, sequential dependencies could be interpreted as a special class of speed constraints declared on time series with fixed time intervals. Moreover, [45] repairs with the variance constraints in a window.

However, with any constraint, the constraint-based repairing identifies and repairs only the violations to the constraints, without indicating the most likely answers among all valid repairs that satisfy the constraints.

8.2 Statistic-based repairing

Correcting data with the statistical features extracted from data is an important direction of data repairing. Some works repair the rational data with the statistical features [23, 28, 44]. Due to the difference between relational data and our studied sequential data, the above methods are not applicable.

Similarly, some statistic-based methods seek data features from time series. IMR [49] builds ARX models from a small amount of labeled data. With unlabeled time sequences, our proposal models the probability of speed changes and obtains accurate repair results by further revealing the probability of repairs.

8.3 Stream processing

Stream processing is necessary in edge computing [43], real-time applications [37] and other aspects. Some frameworks are developed to work for this scenario [2, 29], which focus on distributed systems, high performance, and high reliability. Besides, some works design specialized incremental algorithms [35] or use traditional incremental algorithms to solve the specific problems [39]. Inspired by these ideas, we

Table 7 The confusion matrix over GPS

	TP rate	TN rate	FP rate	FN rate	Precision	Recall	F-Score
Adaptive-DP	0.0197	0.9243	0.0460	0.0100	0.2995	0.6629	0.4126
Adaptive-DPC	0.0143	0.9657	0.0047	0.0153	0.7544	0.4831	0.5890
Adaptive-UG	0.0123	0.9673	0.0030	0.0173	0.8043	0.4157	0.5481
SCREEN	0.0023	0.9697	0.0007	0.0273	0.7778	0.0787	0.1429
SMA [7]	0.0263	0.3497	0.6207	0.0033	0.0407	0.8876	0.0778

design streaming algorithms based on scroll or sliding windows, which achieve linear computing efficiency with limited memory space.

9 Conclusions

In this study, we extend the conference version which focuses on repairing as a whole to the streaming scenario. We capture the challenges of repairing stream data and propose (1) a method of dynamic probability construction, (2) a solution of adaptive budget determination, and (3) streaming repairing algorithms based on scroll and sliding windows. Experiments on several real datasets demonstrate the better performance of our proposal, in both repairing and application accuracy, compared to the state-of-the-art constraint-based repairing.

Acknowledgements This work is supported in part by the National Natural Science Foundation of China (62072265, 62102023, 62021002, 62232005), National Key Research and Development Plan (2021YFB3300500, 2019YFB1705301), Beijing National Research Center for Information Science and Technology (BNR2022RC01011), and Alibaba Group through Alibaba Innovative Research (AIR) Program. Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

References

1. Anderson, C.: The Long Tail. Harper Collins, USA (2008)
2. ASF: Apache storm (2020). http://storm.apache.org/
3. Berger, V.W., Zhou, Y.: Kolmogorov–Smirnov test: Overview. Statistics reference online, Wiley statsref (2014)
4. Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., Seidl, T.: MOA: massive online analysis, a framework for stream classification and clustering. In: Proceedings of the First Workshop on Applications of Pattern Analysis, WAPA 2010, Cumberland Lodge, Windsor, UK, Sept 1–3, 2010, *JMLR Proceedings*, vol. 11, pp. 44–50. JMLR.org (2010)
5. Blázquez-García, A., Conde, A., Mori, U., Lozano, J.A.: A review on outlier/anomaly detection in time series data. ACM Comput. Surv. **54**(3), 56:1-56:33 (2021)
6. Bohannon, P., Flaster, M., Fan, W., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2005, Baltimore, Maryland, USA, June 14-16, 2005, pp. 143–154. ACM (2005)
7. Brillinger, D.R.: Time Series: Data Analysis and Theory, vol. 36. Siam (2001)
8. Candanedo, L.M., Feldheim, V.: Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. Energy Build (2016)
9. Cheung, Y.W., Lai, K.S.: Lag order and critical values of the augmented dickey-fuller test. J. Bus. Econ. Stat. **13**(3), 277–280 (1995)
10. Chi, Y., Wang, H., Yu, P.S., Muntz, R.R.: Moment: maintaining closed frequent itemsets over a stream sliding window. In: Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), 1–4 Nov 2004, Brighton, UK, pp. 59–66. IEEE Computer Society (2004)
11. Dasu, T., Loh, J.M.: Statistical distortion: consequences of data cleaning. Proc. VLDB Endow. **5**(11), 1674–1683 (2012)
12. Ding, Z., Fei, M.: An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. IFAC Proc. Vol. **46**(20), 12–17 (2013)
13. Draper, N.R., Smith, H.: Applied Regression Analysis. Wiley Series in Probability and Mathematical Statistics, 2nd edn. Wiley (1981)
14. Fang, C., Song, S., Mei, Y.: On repairing timestamps for regular interval time series. Proc. VLDB Endow. **15**(9), 1848–1860 (2022)
15. Gama, J., Medas, P., Castillo, G., Rodrigues, P.P.: Learning with drift detection. In: Advances in Artificial Intelligence—SBIA 2004. In: 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, Sept 29–Oct 1, 2004, Proceedings, Lecture Notes in Computer Science, vol. 3171, pp. 286–295. Springer (2004)
16. Gardner, E.S., Jr.: Exponential smoothing: the state of the art-part ii. Int. J. Forecast. **22**(4), 637–666 (2006)
17. Golab, L., Karloff, H.J., Korn, F., Saha, A., Srivastava, D.: Sequential dependencies. Proc. VLDB Endow. **2**(1), 574–585 (2009)
18. Golab, L., Özsu, M.T.: Processing sliding window multi-joins in continuous queries over data streams. In: Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, Sept 9–12, 2003, pp. 500–511. Morgan Kaufmann (2003)
19. Gu, J., Li, W., Cai, X.: The effect of the forget-remember mechanism on spreading. Eur Phys J B **62**(2), 247–255 (2008)
20. Hyndman, R.J., Athanasopoulos, G.: Forecasting: Principles and Practice. OTexts (2018)
21. Jeffery, S.R., Garofalakis, M.N., Franklin, M.J.: Adaptive cleaning for RFID data streams. In: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12–15, 2006, pp. 163–174. ACM (2006)
22. Karp, R.M.: Reducibility among combinatorial problems. In: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972)
23. Krishnan, S., Wang, J., Wu, E., Franklin, M.J., Goldberg, K.: Activeclean: Interactive data cleaning for statistical modeling. Proc. VLDB Endow. **9**(12), 948–959 (2016)

24. Li, X., Dong, X.L., Lyons, K., Meng, W., Srivastava, D.: Truth finding on the deep web: Is the problem solved? Proc. VLDB Endow. **6**(2), 97–108 (2012)
25. Liu, M., Li, M., Golovnya, D., Rundensteiner, E.A., Claypool, K.T.: Sequence pattern query processing over out-of-order event streams. In: Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009–April 2 2009, Shanghai, China, pp. 784–795. IEEE Computer Society (2009)
26. Livshits, E., Kimelfeld, B., Roy, S.: Computing optimal repairs for functional dependencies. ACM Trans. Database Syst. **45**(1), 4:1-4:46 (2020)
27. Ma, Q., Gu, Y., Lee, W., Yu, G., Liu, H., Wu, X.: REMIAN: real-time and error-tolerant missing value imputation. ACM Trans. Knowl. Discov. Data **14**(6), 77:1-77:38 (2020)
28. Mayfield, C., Neville, J., Prabhakar, S.: ERACER: a database approach for statistical inference and data cleaning. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6–10, 2010, pp. 75–86. ACM (2010)
29. Neumeyer, L., Robbins, B., Nair, A., Kesari, A.: S4: distributed stream computing platform. In: ICDMW 2010, The 10th IEEE International Conference on Data Mining Workshops, Sydney, Australia, 13 Dec 2010, pp. 170–177. IEEE Computer Society (2010)
30. Qi, Z., Wang, H., Wang, A.: Impacts of dirty data on classification and clustering models: an experimental evaluation. J. Comput. Sci. Technol. **36**(4), 806–821 (2021). https://doi.org/10.1007/s11390-021-1344-6
31. Song, K.S.: Circuit for generating a scroll window signal in digital image apparatus (1992)
32. Song, S., Cao, Y., Wang, J.: Cleaning timestamps with temporal constraints. Proc. VLDB Endow. **9**(10), 708–719 (2016)
33. Song, S., Li, C., Zhang, X.: Turn waste into wealth: On simultaneous clustering and cleaning over dirty data. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, Aug 10–13, 2015, pp. 1115–1124. ACM (2015)
34. Song, S., Zhang, A.: Iot data quality. In: CIKM'20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, Oct 19–23, 2020, pp. 3517–3518. ACM (2020)
35. Song, S., Zhang, A., Wang, J., Yu, P.S.: SCREEN: stream data cleaning under speed constraints. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2015, Melbourne, Victoria, Australia, May 31–June 4, 2015, pp. 827–841. ACM (2015)
36. Stisen, A., Blunck, H., Bhattacharya, S., Prentow, T.S., Kjærgaard, M.B., Dey, A.K., Sonne, T., Jensen, M.M.: Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In: Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems, SenSys 2015, Seoul, South Korea, Nov 1–4, 2015, pp. 127–140. ACM (2015)
37. Ulm, G., Smith, S., Nilsson, A., Gustavsson, E., Jirstrand, M.: OODIDA: on-board/off-board distributed real-time data analytics for connected vehicles. Data Sci. Eng. **6**(1), 102–117 (2021)
38. Vorburger, P., Bernstein, A.: Entropy-based concept shift detection. In: 6th International Conference on Data Mining (ICDM'06), pp. 1113–1118. IEEE (2006)
39. Wang, H., Chen, S., Gong, W.: Mobility improves accuracy: Precise robot manipulation with COTS RFID systems. In: 19th IEEE International Conference on Pervasive Computing and Communications, PerCom 2021, Kassel, Germany, March 22–26, 2021, pp. 1–10. IEEE (2021)
40. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning structured event logs: a graph repair approach. In: 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13–17, 2015, pp. 30–41. IEEE Computer Society (2015)
41. Wang, J., Song, S., Zhu, X., Lin, X.: Efficient recovery of missing events. Proc. VLDB Endow. **6**(10), 841–852 (2013)
42. Wang, J., Wang, J., Guo, Y.: Scroll-window recursive subspace identification methods for closed-loop system based on orthogonal projection. Inf. Control **43**(1), 56–62 (2014)
43. Xhafa, F., Kilic, B., Krause, P.: Evaluation of iot stream processing at edge computing layer for semantic data enrichment. Future Gener. Comput. Syst. **105**, 730–736 (2020)
44. Yakout, M., Berti-Équille, L., Elmagarmid, A.K.: Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22–27, 2013, pp. 553–564. ACM (2013)
45. Yin, W., Yue, T., Wang, H., Huang, Y., Li, Y.: Time series cleaning under variance constraints. In: Database Systems for Advanced Applications—DASFAA 2018 International Workshops: BDMS, BDQM, GDMA, and SeCoP, Gold Coast, QLD, Australia, May 21–24, 2018, Proceedings, Lecture Notes in Computer Science, vol. 10829, pp. 108–113. Springer (2018)
46. Yu, Y., Zhu, Y., Li, S., Wan, D.: Time series outlier detection based on sliding window prediction. Math. Probl. Eng. **2014** (2014)
47. Yuan, H., Li, G.: A survey of traffic prediction: from spatio-temporal data to intelligent transportation. Data Sci. Eng. **6**(1), 63–85 (2021). https://doi.org/10.1007/s41019-020-00151-z
48. Zhang, A., Song, S., Wang, J.: Sequential data cleaning: a statistical approach. In: Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26–July 01, 2016, pp. 909–924. ACM (2016)
49. Zhang, A., Song, S., Wang, J., Yu, P.S.: Time series data cleaning: from anomaly detection to anomaly repairing. Proc. VLDB Endow. **10**(10), 1046–1057 (2017)