Multivariate Time Series Cleaning under Speed Constraints

Anonymous Author(s)

ABSTRACT

Errors are common in time series due to unreliable sensor measurements. Existing methods focus on univariate data but do not utilize the correlation between dimensions. Cleaning each dimension separately may lead to a less accurate result, as some errors can only be identified in the multivariate case. We also point out that the widely used minimum change principle is not always the best choice. Instead, we try to change the smallest number of data to avoid a significant change in the data distribution. In this paper, we propose MTCSC, the constraint-based method for cleaning multivariate time series. We formalize the repair problem, propose a linear-time method to employ online computing, and improve it by exploiting data trends. We also support adaptive speed constraint capturing. We analyze the properties of our proposals and compare them with SOTA methods in terms of effectiveness, efficiency versus error rates, data sizes, and applications such as classification. Experiments on real datasets show that MTCSC can have higher repair accuracy with less time consumption. Interestingly, it can be effective even when there are only weak or no correlations between the dimensions.

ACM Reference Format:
Anonymous Author(s). 2024. Multivariate Time Series Cleaning under Speed Constraints. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Outliers are common in time series and can have two different meanings (errors or anomalies) [5]. We focus on errors that result from measurement inaccuracies, data collection or system malfunctions [2]. Even in areas such as stocks and flights, a surprisingly large amount of inconsistent data is observed [25]. It is important to identify and correct these erroneous values to ensure the accuracy and reliability of time series analyzes [21].

1.1 Motivation

Constraint-based methods for cleaning time series have been proposed in recent years. Existing studies [32, 33] consider the constraints of speeds and accelerations on value changes, namely speed/acceleration constraints, respectively. They recognize the violations and modify these dirty points according to the minimum change principle [6] to the repaired results that correspond to the defined constraints. However, such a principle leads to maximum/minimum

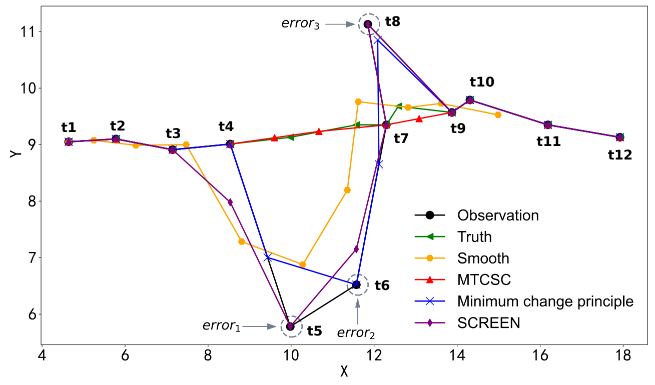

Figure 1: Example of observations and different repairs

compatible values being the final results (referred to as border repair), which can significantly change the data distribution [12]. On the other hand, current constraint-based methods all focus on univariate time series, i.e. with only one dimension. A multivariate data point can meet the given constraint in each of the single dimensions, but still violate the speed constraint in the multivariate case (see data point at t_8 in Example 1.1). Furthermore, they cannot recognize the small errors that actually meet the speed constraints. The small errors are very important in some applications such as automatic driving [38].

Smoothing methods such as [16] use the weighted average of previous values as repaired results and suffer from the problem of over-repair, where almost all data points are changed even though most of them are originally correct. The statistical-based method [38] builds a probability distribution model of speed changes between neighboring data points over the entire data to repair small errors. [36] develops algorithms in streaming scenarios using a dynamic probability construction. Although small errors are successfully repaired, the above methods are still proposed for univariate time series and do not capture the correlations between data dimensions. The more recent learning-based methods [35, 39] pay more attention to the anomalies (not errors), i.e. the observations that deviate significantly from the majority of the data. These methods always require a large number of clean data samples and are quite sensitive to the hyper-parameters.

Example 1.1. Consider a continuous sequence of 12 GPS trajectories in Figure 1. There are considerable deviations in the observations (in black) from t_5, t_6, t_8, which are influenced by the passage through the building and are regarded as errors.

The smoothing method (in orange) changes almost all points, even the originally correct ones. As a univariate method, SCREEN considers each dimension separately. The point at t_8 satisfies the speed constraint in every single dimension and remains unchanged, the point at t_4 is changed accordingly (in purple).

Then we consider using speed constraint in the multivariate case. Following the minimum change principle, the border repair is applied at t_5 and t_8, and t_6 is therefore considered correct. If we take advantage of the data distribution (trend of succeeding points) and manage to change the least number of points, the repair result (our proposal MTCSC, in red) is closer to the ground truth (in green).

1.2 Solution

To overcome the above problems of the existing methods, we apply the speed constraint over all dimensions together instead of considering each dimension separately. Moreover, we do not follow the *minimum change principle* [6] to minimize the total distance before and after the repair, since a certain portion of originally clean data is still changed. Instead, we alternatively manage to fix the minimum number of data points and maintain the data distribution, a.k.a. the *minimum fix principle* [1]. For small errors that are difficult to detect by the speed constraint, we use the information about the data distribution in the sliding window and modify the points even if they satisfy the given constraints. Based on the above techniques, we propose MTCSC to clean the multivariate time series under speed constraints.

However, there still exists some challenges. (1) It may be difficult to explain why it makes sense to consider the speed constraint across all dimensions together. For a GPS trajectory, it is obvious that the speed should be calculated across all two dimensions. But it is not convincing enough to calculate the speed across the dimensions without any correlations. Will this work in practice and why? (2) When considering the multivariate case, the search for a suitable repair candidate is not trivial due to the *curse of dimensionality* [34]. On the one hand, the higher the dimensions, the sparser the data points are, which makes it difficult to set a suitable speed constraint. On the other hand, the candidate space becomes quite large. In the univariate case, the repair candidate lies on a line, in the two-dimensional case on a circle, in the three-dimensional case on a cube, and so on. (3) Time series are not stationary [14] in some scenarios, but the success of our proposal depends on the correct specification of the speed constraint. How to dynamically determine the speed constraint when points arrive becomes another challenge.

1.3 Contribution

The proposed MTCSC is a linear time, constant space cleaning method over multivariate time series. Our major contributions in this paper are summarized as:

(1) We formalize the repair problem over multivariate time series under speed constraints by considering the whole time series or data points in a window, respectively, in Section 2. By transforming to the MIQP/MILP problem, existing solver can be employed. We further reduce the time complexity to polynomial time by using the idea of dynamic programming (MTCSC-G).
(2) We design an online linear time cleaning method (MTCSC-L) in Section 3.1 to locally determine whether the first point (key point) in the current window to be repaired or not as data point arrives. It is notable that soundness w.r.t. speed constraint satisfaction is guaranteed in the devised algorithm.
(3) We enhance the online method by capturing the data distribution in the local window via clustering (MTCSC-C) in Section 3.2 to further promote the accuracy. Considering the trend of the following data points in the given window, small errors that satisfy the speed constraint can also be repaired.
(4) We present an adaptive method (MTCSC-A) that can capture the proper speed constraint in Section 4. Once the difference of the speed distribution in a certain period is larger than a per-defined threshold, which means the characteristic of the series has been changed, the speed constraint is then updated to capture the feature of the incoming data.
(5) We analyze the experimental results on real datasets and discuss the superiority and limitations of our proposed methods in Section 5.

Table 1 lists the notations frequently used in this paper.

Table 1: Notations

Symbol	Description
$\boldsymbol{x}$	time series
$\boldsymbol{x}_i, \boldsymbol{x}[i]$	i-th data point in $\boldsymbol{x}$
t_i	timestamp of i-th data point in $\boldsymbol{x}$
s	speed constraint
w	window size of speed constraint
n	length of time series $\boldsymbol{x}$
D	dimension of time series $\boldsymbol{x}$
$\boldsymbol{x}'$	repair of time series $\boldsymbol{x}$
z_i	0 if $\boldsymbol{x}_i$ is not modified, otherwise, 1
M	a constant number which is sufficient large
satisfy$(\boldsymbol{x}_i, \boldsymbol{x}_j)$	$\boldsymbol{x}_i$ is compatible with $\boldsymbol{x}_j$ w.r.t. s

2 PROBLEM STATEMENT

In this section, we first introduce the time series and the speed constraint that are considered in our proposal. Then, we define the repair problem using the *minimum fix principle* with all data points as a whole. Finally, we formalize the problem when we consider the constraints only locally in a given window.

2.1 Speed Constraint

Definition 2.1 (Time Series). A time series is a sequence of data points indexed in time order. Consider a time series of n observations, $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. The i-th observation (data point) $\boldsymbol{x}_i \in \mathbb{R}^D$ consists of D dimensions $\{x_i^1, \ldots, x_i^D\}$ and is observed at time t_i.

Definition 2.2 (Distance). The distance between two data points is the Euclidean distance between them, denoted by $d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{\ell=1}^{D}(x_i^\ell - x_j^\ell)^2}$.

Definition 2.3 (Speed Constraint). A *speed constraint* $s = (s_{\min}, s_{\max})$ with window size w is a pair of minimum speed $s_{\min}$ and maximum speed $s_{\max}$ over the time series $\boldsymbol{x}$. We say that a time series $\boldsymbol{x}$ *satisfies* the speed constraint s, denoted by $\boldsymbol{x} \vDash s$, if for any $\boldsymbol{x}_i, \boldsymbol{x}_j$ in a window, i.e., $0 < t_j - t_i \le w$, it has $s_{\min} \le \frac{d(\boldsymbol{x}_i, \boldsymbol{x}_j)}{t_j - t_i} \le s_{\max}$, also denoted by satisfy$(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where window w denotes a period of time.

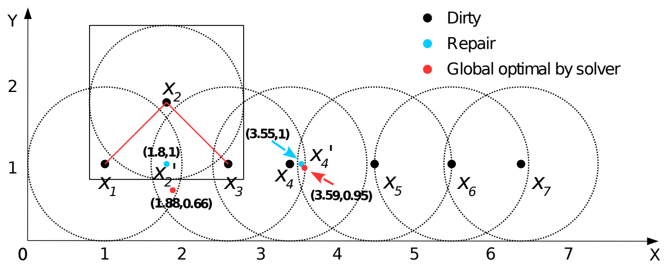

Figure 2: Possible repairs under speed constraints

As mentioned in [33], speed constraint is often useful in real-world scenarios within a certain time period. For example, it makes sense to consider the maximum walking/running/driving speed in hours, as a person cannot usually perform these activities for days or longer without interruption. Stock prices are limited to a certain range based on the last trading day's prices. Speed constraint is applicable in practice. It supports online cleaning and takes less time while maintaining relatively good repair accuracy. Time series is naturally infinite. An online method can better meet these requirements. Meanwhile, speed constraint can be efficiently captured dynamically, which accommodates the non-stationary property of time series. Moreover, it is easy to know that $s_{\min} = 0$ in real cases because the minimum Euclidean distance will always be 0. Therefore, we assume that $s_{\min} = 0$ and refer to $s_{\max}$ as s in the rest of this article.

Another observation is that the "jump" of values in time series data within a short time period is usually limited and has been verified for univariate data (i.e. only one dimension) [32, 33]. As for the multivariate case, one can extend this observation to data that exhibit high correlations over the distance between their dimensions. For example, the GPS trajectory data. However, it is surprising that our proposed speed constraint over multiple dimensions can also be effective for data with weak or no correlations. See Section 5.4.2 for an experimental analysis.

A repair $\boldsymbol{x}'$ of $\boldsymbol{x}$ is a modification of the values in $\boldsymbol{x}_i$ to $\boldsymbol{x}'_i$, i.e., $\exists \ell \in (1, D), x_i^{\ell} \neq x_i^{\ell'}$, where $t'_i = t_i$. Referring to the minimum fix principle, the repair number is counted as

$$\Delta(\boldsymbol{x}', \boldsymbol{x}) = \sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}'_i \neq \boldsymbol{x}_i) = \sum_{i=1}^{n} z_i \tag{1}$$

Example 2.4 (Speed constraints and repairs). Consider a time series $(D = 2)$ $\boldsymbol{x} = \{(1,1), (1.8,1.8), (2.6,1), (3.4,1), (4.5,1), (5.5,1), (6.4,1)\}$ of 7 points, with timestamps $t = \{1,2,3,4,5,6,7\}$. Let the speed constraint $s = 1$. Figure 2 illustrates the data points (in black). The dashed circles in the figure all have a radius of $1 (= s \cdot 1)$. If data points from adjacent timestamps fall into each other's dashed circles, this means that they mutually satisfy the speed constraints.

Let windows size $w = 1$. Points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, with time interval $2 - 1 \leq 1$ in a window, are identified as violations to s, since the speed is $\sqrt{(1.8-1)^2 + (1.8-1)^2}/(2-1) \approx 1.13 > 1$. Similarly, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ with speed $\sqrt{(2.6-1.8)^2 + (1-1.8)^2}/(2-1) \approx 1.13 > 1$ are violations. However, if the two dimensions are considered separately, $\boldsymbol{x}_2$ will be compatible with $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ (denoted by black square, the speed are both 0.8 in x and y direction).

To fix the violations (indicated by red lines), a repair can be performed on $\boldsymbol{x}_2$, i.e.,$\boldsymbol{x}'_2 = (1.8,1)$ (the blue point). As shown in Figure 1, $\boldsymbol{x}'_2$ is compatible with $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$. Similarly, a repair can be performed for $\boldsymbol{x}_4$, i.e. $\boldsymbol{x}'_4 = (3.55,1)$ (the blue point). The repaired sequence therefore satisfies the speed constraints. The repaired points are $\boldsymbol{x}_2$ and $\boldsymbol{x}_4$, and the repair number is 2.

2.2 Global Optimal

The cleaning problem is to find a repair time series, termed as *global optimal*, that satisfies the given speed constraints and minimizes the number of points changed from the original time series.

Problem 1. *Given a finite time series $\boldsymbol{x}$ of n data points and a speed constraint s, the global optimal repair problem is to find a repair $\boldsymbol{x}'$ such that $\boldsymbol{x}' \vDash s$ and $\Delta(\boldsymbol{x}', \boldsymbol{x})$ is minimized.*

The global optimal repair problem can be formalized as follows,

$$\min \quad \sum_{i=1}^{n} z_i, \qquad z_i \in \{0,1\} \tag{2}$$

$$\text{s.t.} \quad \frac{d(\boldsymbol{x}'_i, \boldsymbol{x}'_j)}{t_j - t_i} \leq s, \qquad t_i < t_j \leq t_i + w, 1 \leq i, j \leq n \tag{3}$$

$$x_i^{\ell'} \in [x_i^{\ell} \pm M \cdot z_i], \qquad M > c, 1 \leq i \leq n, 1 \leq \ell \leq D \tag{4}$$

where c is constant integer that large enough to make x_i^{ℓ} feasible.

The correctness of result x' is easy to prove. Formula (2) is the repair number in formula (1) to minimize. The speed constraints are ensured in formula (3), by considering all the t_j in the window starting from t_i. Finally, formula (4) identifies the repair value for each data point i in the time series.

2.2.1 Optimization Solver. After formalization, the problem is a standard mixed-integer quadratic program (MIQP) problem [8], and $z_i, x_i^{\ell'}, 1 \leq i \leq n, 1 \leq \ell \leq D$ are variables in problem solving. If the input is univariate, the problem is reduced to a mixed-integer linear program (MILP) problem [17], since the distance is linear. Existing solvers like Gurobi [19] can be used directly. Once the input time series has been converted to the constraint type, according to the formulas (3) and (4), where M is set to 1000, it can be fed directly into an existing optimization solver. The output of the solver is the repair values of each data points and can be the "optimal" solution under our problem definition. However, the choice of vanilla output as repaired results can still be improved, as we only minimize the number of repaired points but do not consider the magnitude of the changes, which can lead to a repair that is too far from the original point and less accurate.

Example 2.5 (Global optimal by solver, Example 2.4 continued). Consider again the time series $\boldsymbol{x}$ and the speed constraint $s = 1$ in Example 2.4 with window size $w = 7$, as illustrated in Figure 1. According to formula (3), the constraint declared w.r.t. s are:

$$\frac{d(x'_1, x'_2)}{2-1} \leq 1, \quad \frac{d(x'_1, x'_3)}{3-1} \leq 1, \quad \frac{d(x'_1, x'_4)}{4-1} \leq 1, \quad \frac{d(x'_1, x'_5)}{5-1} \leq 1,$$

$$\ldots$$

$$\frac{d(x'_4, x'_7)}{7-4} \leq 1, \quad \frac{d(x'_5, x'_6)}{6-5} \leq 1, \quad \frac{d(x'_5, x'_7)}{7-5} \leq 1, \quad \frac{d(x'_6, x'_7)}{7-6} \leq 1.$$

We use Gurobi Optimizer to get the global optimal solution with the minimum fix is $\boldsymbol{x}'_2 = (1.88, 0.66), \boldsymbol{x}'_4 = (3.59, 0.95)$, as shown by the red points in Figure 2. The repair number is 2.

Algorithm 1: MTCSC-G($\boldsymbol{x}$, s)

```
Input: time series $x$, speed constraint $s$
Output: index of points that need to be fixed $FixList$
1  $dp[1..n] \leftarrow \{1\}, preIndex[1..n] \leftarrow \{0\}$;
2  $maxLength \leftarrow 0, endIndex \leftarrow 0$;
3  for $i \leftarrow 0$ to $n-1$ do
4      for $j \leftarrow 0$ to $i-1$ do
5          if $satisfy(x_i, x_j)$ and $dp[i] < dp[j]+1$ then
6              $dp[i] \leftarrow dp[j] + 1$;
7              $preIndex[i] \leftarrow j$;
8      if $dp[i] > maxLength$ then
9          $maxLength \leftarrow dp[i]$;
10         $endIndex \leftarrow i$;
11 for $k \leftarrow 1$ to $maxLength$ do
12     $CleanList.add(endIndex)$;
13     $endIndex \leftarrow preIndex[endIndex]$;
14 $FixList \leftarrow \{1, \dots n\} - CleanList$;
15 return $FixList$;
```

2.2.2 Dynamic Programming. The optimization solver can lead to a global optimal solution, but it costs too much time. Therefore, we propose Algorithm 1 to determine which data points need to be fixed to improve efficiency. It is an extension of the longest increasing subsequence problem (LIS) [31]. Our goal is to find the longest subsequence in the entire time series that satisfies the given speed constraint, which is equivalent to the LIS problem of finding the longest increasing subsequence. Once it is found (i.e. *CleanList*), the data points that are not included in this list must be repaired and are guaranteed to be the least modified (i.e. minimum fix). Finally, for each point in *FixList*, we can easily find the nearest preceding point $\boldsymbol{x}_p$ and the nearest succeeding point $\boldsymbol{x}_m$ in *CleanList*. The interpolation of these two points according to the formula (6) is used as the final repair result. The correctness can be verified in Proposition 3.1 and Proposition 3.4 in Section 3. As for the time analysis, we know that the judgment procedure satisfy($\boldsymbol{x}_i, \boldsymbol{x}_j$) will cost $O(D)$ time where D is the dimension of time series. Algorithm 1 outputs the *FixList*, which is in fact the detection of violations and the detection time is $O(Dn^2)$. The overall time complexity is $O(Dn^2)+O(fD) = O(Dn^2)$ where f is the number of all the points need to be fixed and the worst-case scenario is $f = n/2$.

Example 2.6 (Global optimal by DP, Example 2.4 continued). Consider again the time series $\boldsymbol{x}$ and the speed constraint $s = 1$ in Example 2.4 with window size $w = 7$. We use Algorithm 1 and get the FixList is {2,4}. This means that the minimum fix repair requires the repair of x_2 and x_4. The clean list is therefore {1,3,5,6,7}. We use the "clean" points (considered clean by Algorithm 1) closest to the detected points on both sides for repair. So we use $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ to repair $\boldsymbol{x}_2$, similarly, we use $\boldsymbol{x}_3$ and $\boldsymbol{x}_5$ to repair $\boldsymbol{x}_4$. The final repaired points are:

$$x_2^{1'} = \frac{t_2 - t_1}{t_3 - t_1}(2.6-1)+1 = 1.8, \qquad x_2^{2'} = \frac{t_2 - t_1}{t_3 - t_1}(1-1)+1 = 1,$$
$$x_4^{1'} = \frac{t_4 - t_3}{t_5 - t_3}(4.5-2.6)+2.6 = 3.55, \qquad x_4^{2'} = \frac{t_4 - t_3}{t_5 - t_3}(1-1)+1 = 1.$$

The global optimal solution with Algorithm 1 with the minimum fix is $\boldsymbol{x}'_2 = (1.8, 1), \boldsymbol{x}'_4 = (3.55, 1)$, shown as blue points in Figure 2. The repair number is 2, which matches the results of the solver.

2.3 Local Optimal

The global optimal requires the entire time series as input and cannot support online cleaning. Therefore, we first study the local optimal, which concerns only the constraints locally in a window. We say a data point $\boldsymbol{x}_k$ *locally satisfies* the speed constraint s, denoted by $\boldsymbol{x}_k \vDash s$, if for any $\boldsymbol{x}_i$ in the window starting from $\boldsymbol{x}_k$, i.e., $t_k < t_i \le t_k + w$, it has satisfy$(\boldsymbol{x}'_i, \boldsymbol{x}'_k)$.

Problem 2. *Given a data point $\boldsymbol{x}_k$ in a time series $\boldsymbol{x}$ and a speed constraint s, the local optimal repair problem is to find a repair $\boldsymbol{x}'$ such that $\boldsymbol{x}'_k \vDash s$ and $\Delta(\boldsymbol{x}', \boldsymbol{x})$ is minimized.*

We formalize the local optimal repair problem as

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} z_i, & & z_i \in \{0,1\} \qquad (5)\\ \text{s.t.} \quad & \frac{d(\boldsymbol{x}'_k, \boldsymbol{x}'_i)}{t_k - t_i} \le s, & & t_k < t_i \le t_k + w, 1 \le i \le n \\ & x_i^{\ell'} \in [x_i^{\ell} \pm M \cdot z_i], & & M > c, t_k < t_i \le t_k + w, 1 \le \ell \le D \end{aligned}$$

Similarly, $z_i, x_i^{\ell'}, 1 \le i \le n, 1 \le \ell \le D$ are variables in problem solving in the existing solver. We only change data points i with $t_k \le t_i \le t_k + w$ in the window starting from the current $\boldsymbol{x}_k$ under the local optimal setting and therefore only have much fewer variables. Nevertheless, the speed constraints are still satisfied in the given window. Moreover, we can also develop a greedy algorithm whose time complexity is $O(n)$. (See Section 3.1 for more details.)

Example 2.7 (Local optimal, Example 2.4 continued). Consider again the time series $\boldsymbol{x}$ and the speed constraint $s = 1$ in Example 2.4 with window size $w = 7$, as illustrated in Figure 1.
Let $t_k = 1$, i.e. $x_1 = (1, 1)$ be the currently considered data point. According to formula (5), the constraint for the local optimal on $\boldsymbol{x}_1$ are:

$$\frac{d(\boldsymbol{x}'_1, \boldsymbol{x}'_2)}{2-1} \le 1, \quad \frac{d(\boldsymbol{x}'_1, \boldsymbol{x}'_3)}{3-1} \le 1, \quad \frac{d(\boldsymbol{x}'_1, \boldsymbol{x}'_4)}{4-1} \le 1,$$
$$\frac{d(\boldsymbol{x}'_1, \boldsymbol{x}'_5)}{5-1} \le 1, \quad \frac{d(\boldsymbol{x}'_1, \boldsymbol{x}'_6)}{6-1} \le 1, \quad \frac{d(\boldsymbol{x}'_1, \boldsymbol{x}'_7)}{7-1} \le 1.$$

In the local optimal, only $\frac{d(\boldsymbol{x}_1, \boldsymbol{x}_2)}{2-1} \approx 1.33 > 1$ violates the speed constraint, while $\frac{d(\boldsymbol{x}_1, \boldsymbol{x}_4)}{4-1} = 2.4 < 3$ satisfies the speed constraint. So we just need to repair $\boldsymbol{x}_2$. We use $\boldsymbol{x}_k$, i.e. $\boldsymbol{x}_1$ and the first data point (i.e. $\boldsymbol{x}_3$) after $\boldsymbol{x}_2$ that satisfies the speed constraint with $\boldsymbol{x}_1$ to repair $\boldsymbol{x}_2$.

$$x_2^{1'} = \frac{t_2 - t_1}{t_3 - t_1} \cdot (2.6-1)+1 = 1.8, \qquad x_2^{2'} = \frac{t_2 - t_1}{t_3 - t_1} \cdot (1-1)+1 = 1.$$

The local optimal solution with the minimum fix is $x'_2 = (1.8, 1)$. The repair number is 1, which is less than the global optimal because the constraints of the local optimal are fewer than those of the global optimal.

3 STREAMING COMPUTATION

A time series can also have an infinite number of data points, i.e. it arrives continuously. To support online cleaning, we first propose MTCSC-L w.r.t. local optimal by deciding whether the current data point should be fixed or not in Section 3.1. To support cleaning errors that satisfy the speed constraint, we further devise MTCSC-C, which takes into account the trend of data points in the current window in Section 3.2. By sliding windows in the time series, both the result of MTCSC-L x_{local} and MTCSC-C x_{clu} guarantee to satisfy the given speed constraints. Since the global optimal provides a minimum fixed repair x_{global} over the entire data, we always have $\Delta(\boldsymbol{x}, x_{\text{global}}) \le \Delta(\boldsymbol{x}, x_{\text{local}})$ and $\Delta(\boldsymbol{x}, x_{\text{global}}) \le \Delta(\boldsymbol{x}, x_{\text{clu}})$. Unfortunately, we do not have a theoretical upper bound of local and cluster fix number compared to the global ones. In practice, the fix numbers are very close to each other (as shown in Figure 5(d)).

3.1 Local Streaming

The proposed MTCSC-L is to iteratively determine the local optimal $\boldsymbol{x}'_k$, for $k = 1, 2, \dots$. Consider $\boldsymbol{x}_k$, where $\boldsymbol{x}'_1, \dots, \boldsymbol{x}'_{k-1}$ have been determined before. Referring to the speed constraints, each fixed $\boldsymbol{x}'_j, t_k - w \le t_j < t_k, 1 \le j < k$, provides a range of candidates for $\boldsymbol{x}'_k$, i.e., $\{(x_k^{1'}, \dots, x_k^{D'}) \mid d(\boldsymbol{x}'_k, \boldsymbol{x}'_j) \le s(t_k - t_j)\}$. The following proposition states that considering the last $\boldsymbol{x}'_{k-1}$ is sufficient to determine the range of possible repairs for $\boldsymbol{x}'_k$. The rationale is that for any $1 \le j < i < k$, $\boldsymbol{x}'_i$ should be in the range specified by $\boldsymbol{x}'_j$ as well. In other words, the candidate range of x'_k specified by x'_i is subsumed in the range by x'_j.

Proposition 3.1. *For any $1 \le j < i < k, t_k - w \le t_j < t_i < t_k$, we have $\{\boldsymbol{x}'_k \mid d(\boldsymbol{x}'_k, \boldsymbol{x}'_i) \le s(t_k - t_i)\} \subset \{\boldsymbol{x}'_k \mid d(\boldsymbol{x}'_k, \boldsymbol{x}'_j) \le s(t_k - t_j)\}$.*

Proof. W.l.o.g. we take the two dimensional data $\boldsymbol{x}_k = \{(x_k, y_k)\}$ for all the proofs. Since $\boldsymbol{x}'_i$ and $\boldsymbol{x}'_j$ are all fixed and in the same window, we have $d(\boldsymbol{x}'_i, \boldsymbol{x}'_j) \le s(t_i - t_j)$. The range specified by $\boldsymbol{x}'_k = \{(x, y) \mid \sqrt{(x - x'_i)^2 + (y - y'_i)^2} \le s(t_k - t_i)\}$, i.e., $d(\boldsymbol{x}'_k, \boldsymbol{x}'_i) \le s(t_k - t_i)$. Points $\boldsymbol{x}'_k, \boldsymbol{x}'_i, \boldsymbol{x}'_j$ can form a triangle or a line. Following the triangle inequality [?], we have $d(\boldsymbol{x}'_k, \boldsymbol{x}'_j) \le d(\boldsymbol{x}'_k, \boldsymbol{x}'_i) + d(\boldsymbol{x}'_i, \boldsymbol{x}'_j) \le s(t_k - t_i) + s(t_i - t_j) = s(t_k - t_j)$. Hence, $\boldsymbol{x}'_k$ is in the range specified by $\boldsymbol{x}'_j$. □

We can therefore identify the repair $\boldsymbol{x}'_k$ in a tight range of candidates specified by $\boldsymbol{x}'_{k-1}$. However, the above ranges are still too large even if the dimension of the data is two (a circle of radius $s(t_k - t_{k-1})$). Fortunately, we find that the sequence of speed changes is stationary for a short period of time [36]. Then we use the idea of interpolation to determine the local solution of x'_k as

$$x_k^{\ell'} = \alpha \cdot (x_m^{\ell} - x_p^{\ell'}) + x_p^{\ell'}, \quad 1 \le \ell \le D \tag{6}$$

where $\alpha = (t_k - t_p)/(t_m - t_p)$, p is the last point before the window, k is the first point in the window (key point) and m is a point after k with satisfy$(\boldsymbol{x}'_p, \boldsymbol{x}_m)$. The following proposition states that such a solution satisfies the given speed constraint.

Proposition 3.2. *For any $\ell = 1, \dots, D$, $x_k^{\ell'} = \alpha \cdot (x_m^{\ell} - x_p^{\ell'}) + x_p^{\ell'}$, we have satisfy$(\boldsymbol{x}'_p, \boldsymbol{x}'_k)$.*

Proof. Let $r_1 = s(t_k - t_p)$, $r_3 = s(t_m - t_p)$ and $\ell \in [1, D]$. Since satisfy$(\boldsymbol{x}'_p, \boldsymbol{x}_m)$, we have $x_m^{\ell} \in [x_p^{\ell'} - r_3, x_p^{\ell'} + r_3]$.

$$\begin{aligned}
x_k^{\ell'} &= \alpha \cdot (x_m^{\ell} - x_p^{\ell'}) + x_p^{\ell'} = \alpha \cdot x_m^{\ell} + (1 - \alpha) x_p^{\ell'} \\
&\ge \alpha \cdot (x_p^{\ell'} - r_3) + (1 - \alpha) x_p^{\ell'} = x_p^{\ell'} - \alpha \cdot r_3 \\
&= x_p^{\ell'} - s(t_m - t_p) \cdot (t_k - t_p)/(t_m - t_p) \\
&= x_p^{\ell'} - s(t_k - t_p) = x_p^{\ell'} - r_1
\end{aligned}$$

Similarly, we have $x_k^{\ell'} \le x_p^{\ell'} + r_1$. Since point $\boldsymbol{x}'_k$ is in the same line with points $\boldsymbol{x}'_p$ and $\boldsymbol{x}_m$, hence, we have satisfy$(\boldsymbol{x}'_p, \boldsymbol{x}'_k)$. □

Algorithm 2 presents the online local repair of a time series $\boldsymbol{x}$ w.r.t. local optimal under the speed constraint s. It detects the violation for each data point in Line 2 and the time cost is $O(D)$. For each point k, $k = 1, 2, \dots, n$ that violates s w.r.t. point $k - 1$, Line 4 will find the first succeeding point i that satisfy$(\boldsymbol{x}_i, \boldsymbol{x}'_{k-1})$. Lines 5 and 6 indicate that if there is no point satisfying s w.r.t. point $k - 1$, $\boldsymbol{x}'_{k-1}$ will be determined as the repair of point k. Otherwise, Lines 8 to 12 will set $\boldsymbol{x}'_k$ following formula (6). It is easy to know that the time complexity of Algorithm 2 is $O(wDn)$.

Algorithm 2: MTCSC-L($\boldsymbol{x}$, s)

```
   Input: time series $\boldsymbol{x}$, speed constraint $s$
   Output: a repair $\boldsymbol{x}'$ of $\boldsymbol{x}$ w.r.t local optimal
1  for $k \leftarrow 1$ to $n$ do
2      if satisfy($\boldsymbol{x}_k, \boldsymbol{x}'_{k-1}$) or $k = 1$ then
3          continue;
4      for $i \leftarrow k + 1$ to $n$ do
5          if $t_i > t_k + w$ then
6              $\boldsymbol{x}'_k \leftarrow \boldsymbol{x}'_{k-1}$;
7              break;
8          if satisfy($\boldsymbol{x}_i, \boldsymbol{x}'_{k-1}$) then
9              $\alpha \leftarrow (t_k - t_{k-1})/(t_i - t_{k-1})$;
10             for $\ell \leftarrow 1$ to $D$ do
11                 $x_k^{\ell'} \leftarrow \alpha \cdot (x_i^{\ell} - x_{k-1}^{\ell'}) + x_{k-1}^{\ell'}$;
12             break;
13 return $\boldsymbol{x}'$;
```

Example 3.3 (Local streaming, Example 2.4 continued). Consider again the time series $\boldsymbol{x}$ and the speed constraints $s = 1$ in Example 2.4 with window size $w = 2$.

Let $t_k = 2$, i.e. $\boldsymbol{x}_2 = (1.8, 1.8)$ be the key point, then the current window is $\{\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4\}$. According to Algorithm 2, since $\boldsymbol{x}_2$ violates the speed constraint with $\boldsymbol{x}'_1 = (1, 1)$, we find the first succeeding point i that satisfy$(\boldsymbol{x}_i, \boldsymbol{x}'_1)$, i.e. $\boldsymbol{x}_3 = (2.6, 1)$. A repair on $\boldsymbol{x}_2$ is :

$$x_2^{1'} = \frac{t_2 - t_1}{t_3 - t_1}(2.6 - 1) + 1 = 1.8, \quad x_2^{2'} = \frac{t_2 - t_1}{t_3 - t_1}(1 - 1) + 1 = 1.$$

When the window is $\{x_5, x_6, x_7\}$, a repair on x_5 is :

$$x_5^{1'} = \frac{t_5 - t_4}{t_7 - t_4}(6.4 - 3.4) + 3.4 = 4.4, \quad x_5^{2'} = \frac{t_5 - t_4}{t_7 - t_4}(1 - 1) + 1 = 1.$$

Similarly, a repair on x_6 is performed, i.e. $x_6' = (5.4, 1)$. Hence, the repair number is 3 and we have $\Delta(x, x_{\text{global}}) = 2 \le \Delta(x, x_{\text{local}}) = 3$.

3.2 Online Clustering

As mentioned before, MTCSC-L uses the first succeeding point m with satisfy(x_p', x_m) to help repairing. However, the correctness of the point m is not guaranteed, i.e. the potential outlier that satisfies the speed constraint can mislead the repair result. Therefore, we propose MTCSC-C to fully utilize the data distribution in the current window. We first show how to determine the candidate range. Then we introduce the method BuildCluster to construct the cluster in the given window and explain the concept behind it. Finally, we describe the whole procedure to obtain the final result.

3.2.1 Candidate Range. Proposition 3.1 shows that we can always have a tighter range specified by the nearest preceding point. In addition, the following proposition shows that this property also applies to succeeding points.

Proposition 3.4. *For any $i > j > k, t_k < t_j < t_i < t_k + w$, if satisfy$(x_j, x_i)$, then we have $\{x_k' \mid d(x_k', x_j) \le s(t_j - t_k)\} \subset \{x_k' \mid d(x_k', x_i) \le s(t_i - t_k)\}$.*

Proof. Referring to the proof of Proposition 3.1, once x_k' is in the range specified by x_i, it is also in the range specified by x_j. □

Hence, together with the assumption in Section 3.1, Proposition 3.1 and Proposition 3.4 show that considering the nearest preceding and succeeding point (adjacent points) is sufficient to determine the final candidate range for the key point k. Since the preceding point p is already fixed, we now turn to find a suitable succeeding point.

3.2.2 Cluster Construction. Algorithm 3 obtains the main trend of the data distribution in the window by clustering with a time complexity of $O(w^2 D)$. Line 2 creates an array that records the flag f for each point: (1) 0, the default value (dirty); (2) -1, which is itself the first point of a cluster; (3) a number greater than 0, the first point of the cluster to which it belongs. Lines 3 to 6 find the first point ℓ that satisfies the speed constraint with the previous fixed point p and create a new cluster containing it. If no points are satisfied, the algorithm skips the for loop in Line 7 and returns an empty list. Other points before ℓ are omitted as they are not in the range specified by the point p. Starting from Line 7, for each point i after ℓ, we check its relationship with its preceding point $j = i - 1$ and do one of the four actions:

Action 1 Add i into the cluster containing j (Lines 9 to 14);
Action 2 Create a new cluster for i (Lines 15 to 18);
Action 3 Continue to check before j ($j = j - 1$) with i (Line 8);
Action 4 Omit i (Line 7).

Explanations. The reasons for Action 1 and Action 4 are obvious, i.e. we try to add 'clean' points to existing clusters and omit the dirty points, respectively.

In the following, we will discuss the reasons for other actions in detail, with respect to the three cases where Satisfy$(W[i], W[j = i - 1])$ is not satisfied: case 1: i is dirty but j is clean; case 2: i is clean but j is dirty; case 3: i and j are both dirty.

Algorithm 3: BuildCluster(x_p', W, s)

```
Input: the last preceding point $x_p'$, the window $W$
       starting from key point $x_k$, speed constraint $s$
Output: cluster set $Clusters$ formed from $W$ except $x_k$
1  $Map < i, Cluster > map \leftarrow \{\}$;
2  $f[1..W.length] \leftarrow \{0\}$;
3  for $\ell \leftarrow 1$ to $W.length$ do
4      if satisfy$(x_p', W[\ell])$ then
5          $f[\ell] \leftarrow -1, map.put(\ell, new\ Cluster(\ell))$;
6          break;
7  for $i \leftarrow \ell + 1$ to $W.length$ do
8      for $j \leftarrow i - 1$ to $\ell$ do
9          if satisfy$(W[i], W[j])$ then
10             if $f[j] = -1$ then
11                 $f[i] \leftarrow j, Cluster(j).add(i)$;
12             else if $f[j] > 0$ then
13                 $f[i] \leftarrow f[j], Cluster(f[i]).add(i)$;
14             break;
15         else if $j = \ell$ or $f[j] > 0$ then
16             if satisfy$(x_p', W[i])$ then
17                 $f[i] \leftarrow -1, map.put(i, new\ Cluster(i))$;
18             break;
19 return $map.values()$;
```

Action 2 creates a new cluster for the point i if j is not the first point of a cluster ($f[j] > 0$), or all previous points in the window are incompatible, as long as i is compatible with the fixed point p. (1) In case 1, it guarantees that the dirty point i forms a new cluster and does not collide with the existing clean cluster; (2) In case 2, the clean point i is not polluted by the existing dirty cluster; (3) In case 3, dirty points with different properties are separated and the possibility of a dirty cluster becoming the largest is reduced.

Action 3 lets i proceed to check the relation to $j - 1$ if j is the first point of a cluster ($f[j] = -1$) or j has already been omitted ($f[j] = 0$). (1) In cases 1 and 3, i has three types of outcomes: (a) becomes a member of a clean cluster, which, as mentioned above, has no effect on the final choice; (b) becomes a member of a dirty cluster that corresponds to the purpose of collecting data with similar properties; (c) creates a new cluster that is isolated from the others, as it is a dirty point. So moving on is a smart option. (2) In case 2, i continues the search to join the clean cluster.

3.2.3 Repair via Cluster. Algorithm 4 introduces the online cleaning method via clustering. For each point k as the key point, Lines 3 to 7 collect all points in the window starting with k, and Line 8 constructs the clusters. After collecting the data distribution of the given window, Line 9 considers the first point of the largest cluster as the representative for cleaning. Line 10 recognizes whether x_k violates the speed constraint, so that the time complexity of the detection for each point is $O(w^2 D + 2D)$. In Lines 11 to 13 , x_k' is determined according to the formula (6). The time complexity of Algorithm 4 is $O(w^2 Dn)$. As mentioned above, Algorithm 4 is an improvement of Algorithm 2 as it searches for a more "accurate"

Algorithm 4: MTCSC-C($\boldsymbol{x}$, s)

```
Input: time series $\boldsymbol{x}$, speed constraint $s$
Output: a repair $\boldsymbol{x}'$ of $\boldsymbol{x}$
1  $W \leftarrow \{\}$;
2  for $k \leftarrow 2$ to $n$ do
3      $W.add(\boldsymbol{x}_k)$;
4      for $i \leftarrow k+1$ to $n$ do
5          if $t_i > t_k + w$ then
6              break;
7          $W.add(\boldsymbol{x}_i)$;
8      $Clusters \leftarrow$ BuildCluster($\boldsymbol{x}'_{k-1}$, $W$, $s$);
9      $i \leftarrow \arg\max_j Cluster(j).size() + k$;
10     if !(satisfy($\boldsymbol{x}'_{k-1}, \boldsymbol{x}_k$) and satisfy($\boldsymbol{x}_k, \boldsymbol{x}_i$)) then
11         $\alpha \leftarrow (t_k - t_{k-1})/(t_i - t_{k-1})$;
12         for $\ell \leftarrow 1$ to $D$ do
13             $x_k^{\ell'} \leftarrow \alpha \cdot (x_i^{\ell} - x_{k-1}^{\ell'}) + x_{k-1}^{\ell'}$;
14     $W.clear()$;
15 return $\boldsymbol{x}'$;
```

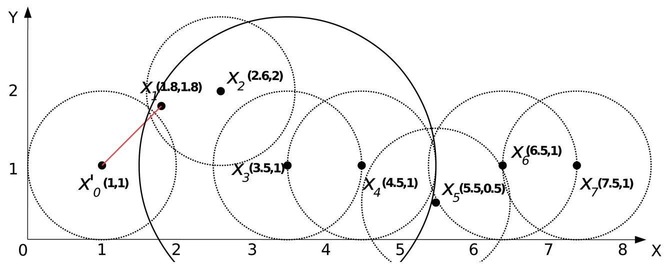

Figure 3: Build Cluster and Repair via Cluster

succeeding point to identify the candidate range. Further discussion on the difference between them can be found in Section 5.

Example 3.5 (Repair via Cluster). Consider a time series (D = 2) $\boldsymbol{x} = \{(1,1), (1.8,1.8), (2.6,2), (3.5,1), (4.5,1), (5.5,0.5), (6.5,1), (7.5,1)\}$ of 8 points, with timestamps $t = \{0,1,2,3,4,5,6,7\}$. Figure 3 shows the data points (in black). We assume that $\boldsymbol{x}_0$ has been repaired. Let the speed constraint $s = 1$ and windows size $w = 6$.

Let $t_k = 1$, i.e. $\boldsymbol{x}_1$ be the key point, then the current window is $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_7\}$ and the previous fixed point is $\boldsymbol{x}'_0$. According to lines 3-6 of Algorithm 3, $\boldsymbol{x}_2$ satisfies the speed constraint with $\boldsymbol{x}'_0$ ($\sqrt{(2.6-1)^2 + (2-1)^2}/(2-0) \approx 0.94 < 1$). So we create a new cluster containing $\boldsymbol{x}_2$.

Next, starting from line 7 of Algorithm 3, we proceed to evaluate different scenarios for the following points. We only show the first two points after $\boldsymbol{x}_2$ due to the space limit.

$\boldsymbol{x}_3$: $\boldsymbol{x}_3$ is not compatible with $\boldsymbol{x}_2$ and there are no data points before $\boldsymbol{x}_2$, but it is compatible with $\boldsymbol{x}'_0$, thus we create a new cluster for $\boldsymbol{x}_3$ (Action 2 and $j = \ell$).

$\boldsymbol{x}_4$: $\boldsymbol{x}_4$ is compatible with $\boldsymbol{x}_3$, we add $\boldsymbol{x}_4$ into the cluster containing $\boldsymbol{x}_3$ (Action 1).

In the end, three clusters will be formed, namely $\{\boldsymbol{x}_2\}$, $\{\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_6, \boldsymbol{x}_7\}$ and $\{\boldsymbol{x}_5\}$. According to Algorithm 4, the repair based on $\boldsymbol{x}'_0$ and $\boldsymbol{x}_3$ for $\boldsymbol{x}_1$ is:

$$x_1^{1'} = \frac{t_1 - t_0}{t_3 - t_0}(3.5-1) + 1 \approx 1.83, \quad x_1^{2'} = \frac{t_1 - t_0}{t_3 - t_0}(1-1) + 1 = 1.$$

The final repair result obtained through Algorithm 3 and Algorithm 4 is $\boldsymbol{x}'_1 = (1.83, 1)$, $\boldsymbol{x}'_2 = (2.66, 1)$ and $\boldsymbol{x}'_5 = (5.5, 1)$.

4 ADAPTIVE SPEED CONSTRAINT

A precise speed constraint is crucial for the performance of all proposed methods. [32, 33] capture the speed constraint via (1) domain experts or common sense; (2) extraction from the data by the 95% confidence level [23]. However, it is still not trivial to set an appropriate speed constraint, especially in the multivariate case, since (1) we may not obtain sufficient domain knowledge, especially if the dimensions are only weakly correlated; (2) the confidence level 95% is not always large enough to set an appropriate speed constraint; (3) the time series (stream) is often non-stationary [14], i.e. the properties of the data may change. In such cases, the predefined speed constraint may take effect in that hour, but may no longer bind the data correctly in the next hour. For example, a person may change their modes of transportation in a short time [40], e.g. walk to the bus stop and then take the bus.

To solve the above problems, we develop an adaptive method to dynamically set an appropriate speed constraint by monitoring the difference in data distribution between adjacent sliding windows.

4.1 Speed Distribution

Theoretically, we should collect a large number of data points to model different phases of an evolving time series. However, it is not possible to obtain such training data for each dataset, as different people and spaces have their own characteristics, e.g. different walking speeds. We therefore intend to use the observations directly to capture the speed distribution.

Another problem arises from the sparsity of time series whose data are continuous. If we want the frequency distribution to work well, we need to collect our data in buckets to get a high enough probability for each category [36]. Then we use the KL divergence [24] to measure the distance between two distributions [12].

4.2 Adaptive Speed

Solution. We propose an online cleaning method MTCSC-A with adaptive speed. Algorithm 5 is the core part and introduces how to set the speed constraint dynamically with a complexity of $O(D + b + m)$. First, we construct the initial speed distribution in the first window W_1 and the second window W_2, where W_1 and W_2 are adjacent and disjoint. W_1 and W_2 store the speed instead of the values of data points to save computation time. When data points arrive, we update the distribution in W_1, W_2 and calculate the distance between them. Once this is greater than a threshold τ, we change the speed constraint with a modify factor β ($s' = s_{95\%}(in\ W_2)/\beta$) and also update the distributions in W_1, W_2 (since the points in the buckets also change). UpdateDistribution(W_1, b, s) represents dividing the speed stored in W_1 into b buckets at equal intervals based on the speed constraint s. MTCSC-A is formed by inserting Algorithm 5 through "s = AdaptiveSpeed($\boldsymbol{x}_{k-1}$, $\boldsymbol{x}_k$, s, b, τ, m, β, W_1, W_2)" between the 3rd and 4th lines of Algorithm 4. Overall, the

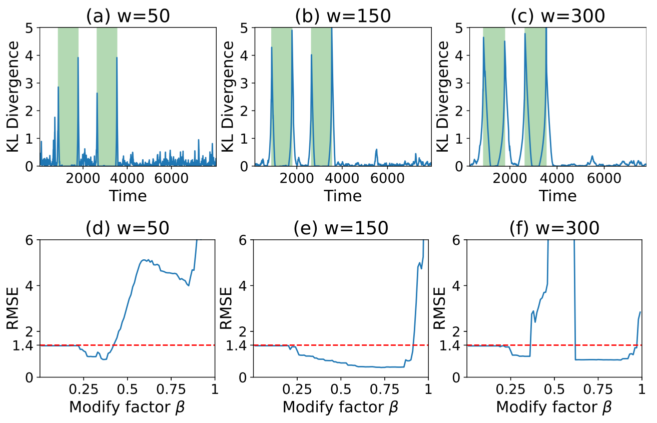

Figure 4: KL divergence/RMSE under different window sizes

time complexity of MTCSC-A is $O((D + b + m + w^2D + 2D) * n) = O(w^2Dn)$.

```
Algorithm 5: AdaptiveSpeed($x_p$, $x_k$, $s$, $b$, $\tau$, $m$, $\beta$, $W_1$, $W_2$)
Input: the last preceding point $x_p$, the key point $x_k$, speed
       constraint $s$, bucket number $b$, threshold $\tau$,
       monitoring interval $m$, modify factor $\beta$, the first
       window $W_1$, the second window $W_2$
Output: speed constraint $s'$
1  $s' = s$, $s_1 \leftarrow d(x_k, x_p)/(t_k - t_p)$;
2  if $W_1.size < m$ then
3      $W_1.push(s_1)$, UpdateDistribution($W_1, b, s$);
4  else if $W_2.size < m$ then
5      $W_2.push(s_1)$, UpdateDistribution($W_2, b, s$);
6  else if $W_2.size = m$ then
7      if $KL(W_1, W_2) > \tau$ then
8          $s' = s_{95\%}(in\ W_2)/\beta$;
9      $s_2 \leftarrow W_2.front()$, $W_2.pop()$;
10     $W_1.push(s_2)$, $W_1.pop()$, UpdateDistribution($W_1, b, s'$);
11     $W_2.push(s_1)$, UpdateDistribution($W_2, b, s'$);
12 return $s'$;
```

Settings. We have the following four hyper-parameters.

- Bucket number b. Figures 15(a) show that b it is not sensitive.
- Threshold τ. As can be seen in Figures 4(a, b, c), there is a clear difference between the KL divergence at the time when the speed changes or not (green blocks indicate a different mode), regardless of the window size.
- Monitoring interval m. Obviously there is a trade-off of m , as shown in Figure 4. (1) For a small $m = 50$, it is difficult to determine the updated s because the speed distribution may not be reliable due to the small number of data points. The red line in Figure 4(d) shows the RMSE of the observations without any repair. We find that only a small interval of β can lead to better performance. (2) On the other hand, $m = 300$ is large enough to collect a number of data points that $s_{95\%}$ is relatively accurate. So if β is small, the updated constraint s is too large to identify the error points. At the same time, the updated constraint must not be too small, as this would mislead the cluster result. However, such a large m also reduces the range in which τ can be selected. We find that 2 is sufficient for $m = 50$, 3 for $m = 150$ and 4 or more for $m = 300$. (3) Intuitively, we would consider a "proper" setting with a moderate value, for example $m = 150$ in the figure. Following this guide, a good m could practically be chosen by observing the width of the KL divergence.
- Modify factor β. It is related to the setting of m and has already been discussed above.

Example 4.1 (Adaptive speed). Let $s = 2.2$ and $b = 6$, the speed bucket is {[0,0.44], (0.44,0.88], (0.88,1.32], (1.32,1.76], (1.76,2.2], (2.2, +∞]}. The last bucket is placed at a point that exceeds the current speed constraint. Consider the speed distribution of the first window $W_1 = \{0, 0, 0, 90, 60, 0\}$ and second window $W_2 = \{3, 4, 1, 44, 25, 73\}$ with $\tau = 0.75$, $m = 150$ and $\beta = 0.75$.

For W_1 and W_2, the probability distribution $P_1 = \{0, 0, 0, 0.6, 0.4, 0\}$ and $P_2 = \{0.02, 0.027, 0.007, 0.293, 0.167, 0.487\}$. The calculation process of KL divergence between distributions P_1 and P_2 is:

$$\begin{aligned}D_{KL}(P_1||P_2) &= 0 \times \log\frac{0}{0.02} + 0 \times \log\frac{0}{0.027} + 0 \times \log\frac{0}{0.007}\\ &\quad + 0.6 \times \log\frac{0.6}{0.293} + 0.4 \times \log\frac{0.4}{0.167} + 0 \times \log\frac{0}{0.487}\\ &= 0 + 0 + 0 + 0.6 \times (0.717) + 0.4 \times (0.873) + 0\\ &= 0.7794\end{aligned}$$

Since the D_{KL} is greater than the threshold τ, s is thus modified to $s_{95\%}(in\ W_2)/\beta = 3.572/0.75 \approx 4.763$. Meanwhile, the speed distribution of W_1 and W_2 will be updated using the latest speed bucket ({[0,0.9526], (0.9526,1.9052], (1.9052,2.8578], (2.8578,3.8104], (3.8104,4.763], (4.763, +∞]}), with $W_1 = \{0, 131, 19, 0, 0, 0\}$ and $W_2 = \{7, 62, 14, 61, 4, 2\}$. There are only two points that exceed the latest speed, indicating that our latest speed settings are relatively accurate.

5 EXPERIMENT

In this section, we use experiments to evaluate performance (1) on univariate time series with injected errors; (2) on multivariate time series with different degrees of correlation and injected errors; (3) on manually collected GPS trajectories (one of them with mode changes) with real errors; and (4) the improvement of data mining tasks such as classification and clustering. Finally, we summarize the highlights and discuss our limitations. The code and data of this work are available online[1].

5.1 Settings

We run experiments on a Windows 10 laptop with a 2.30GHz Intel Core CPU and 32GB RAM. Deep-learning methods are employed with GPU without comparing the efficiency.

5.1.1 Datasets. We use seven public real-world datasets that are originally clean or clean after simple pre-processing with various sizes and dimensions. To analyze the performance, we generate synthetic errors following the guideline in [33]. Errors are injected

[1]https://anonymous.4open.science/r/mtcsc-E4CC

Table 2: Summary of datasets

	Size	#Dim	Error	#Series
Stock [33]	12k	1	Clean	1
ILD [27]	43k	3	Clean after pre-process	1
Tao [3]	568k	3	Clean after pre-process	1
ECG[37]	94k	32	Clean after pre-process	1
GPS(Walk)	11k	2	Embedded	1
GPS(Mixed)	8k	2	Embedded	1
ArrowHead [9]	251	1	Clean	211
AtrialFib [4]	640	2	Clean	30
DSR [9]	345	1	Clean	16
SWJ [4]	2500	4	Clean	27

by randomly replacing the values of some data points in a given dimension. For each replaced data point, it takes a random value between the minimum and maximum values in the dataset. Moreover, we collect two sets of GPS data via smartphones. One contains only walking data, the other is a mixture of walking, running and cycling. The embedded errors are manually labeled according to the exact routes and maps. In order to avoid the effect of randomness, we run experiments on datasets with synthetic errors 10 times with different generating seeds and report the average results. The summary of all datasets is shown in Table 2.

5.1.2 *Metrics.*

- RMSE [22] is employed to evaluate the repair accuracy.
- Repair distance $\delta(\boldsymbol{x}', \boldsymbol{x}) = \sum_{i=1}^{n} d(\boldsymbol{x}'_i, \boldsymbol{x}_i)/n$ presents the total distance between the time series before and after repair.
- Repair number $\Delta(\boldsymbol{x}', \boldsymbol{x}) = \sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}'_i \neq \boldsymbol{x}_i)/n$ shows the extent to which a method changes the input data.
- Time cost. Apart from loading data, error injection and result evaluation, the total time of other procedures is reported.

5.1.3 Baselines. We compare our proposals with 9 competing baselines of different kinds in Table 3. SCREEN and SpeedAcc use the univariate speed constraint and follow the widely used minimum change principle. They are selected to verify the validity of the proposed minimum fix principle. HTD calculates the correlation between the dimensions and performs data cleaning over multivariate time series based on speed constraint. RCSWS aims to clean GPS data and can only implement on two-dimensional data. LsGreedy is a statistical SOTA method that has good accuracy. Due to the inspiration from interpolation in choosing repair solution, we also choose the classical smoothing-based EWMA method as a baseline. Holoclean is a representative of the machine learning method, but it is proposed for relational data. We thus convert time series to relational data, treating timestamps and the names of each dimension as attributes and the values of each dimension as cells. We also convert the speed constraint of each dimension to the form of a denial constraint (which can be recognized by HoloClean) and add the parser module into the source code. We find that there are few works that use deep leaning techniques for time series cleaning. Therefore, we select two SOTA anomaly detection methods, TranAD and CAE-M, and consider their predicted/reconstructed values as repair candidates. We recommend MTCSC-C as the general method and refer to it as MTCSC in the following experiments.

Table 3: Summary of compared methods

Algorithm	Dimension	Process	Type
MTCSC-G	multivariate	batch	constraint
MTCSC-L	multivariate	online	constraint
MTCSC-C	multivariate	online	constraint + statistical
MTCSC-A	multivariate	online	constraint + statistical
SCREEN [33]	univariate	online	constraint
SpeedAcc [32]	univariate	online	constraint
LsGreedy [38]	univariate	online	statistical
EWMA [16]	univariate	online	smoothing
RCSWS[15]	multivariate	online	constraint + statistical
HTD [41]	multivariate	batch	constraint
HoloClean [30]	multivariate	batch	machine learning
TranAD[35]	multivariate	online	deep learning
CAE-M[39]	multivariate	online	deep learning

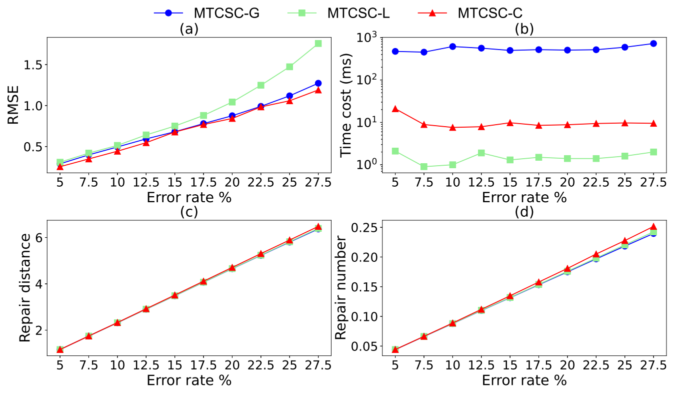

Figure 5: Varying error rate on Stock

5.2 Comparison on Univariate Data

Although we focus on the multivariate case, we also investigate whether we can outperform the univariate specific competitors on univariate time series.

5.2.1 Comparison among our proposals. We compare our proposals on Stock and the results are shown in Figure 5, which verifies our theory in Section 3. Figure 5(a) shows that MTCSC-G and MTCSC-C behave similarly and are better than MTCSC-L. Figure 5(b) shows that MTCSC-G takes the most time, while MTCSC-L takes the least time. MTCSC-C achieves better results than MTCSC-G and takes less than half the time. Figure 5(d) shows that MTCSC-G modifies the fewest points, while both MTCSC-L and MTCSC-C modify slightly more points compared to MTCSC-G.

5.2.2 Varying Error Rate e%. Figure 6(a) shows that the proposed MTCSC achieves the best accuracy and is robust to error rates. When the error rate reaches 20%, the RMSE of LsGreedy suddenly jumps and shows a strong growth trend. The direct reason for this can be found in Figure 6(d), that LsGreedy repairs fewer points then. The deeper insight is that the statistical based method LsGreedy treats the wrong points as correct ones when the error rate is high. Figure 6(b) shows that MTCSC has good efficiency. As shown in Figures 6(c) and (d), we find that SCREEN, SpeedAcc, LsGreedy, and HTD, which are based on the minimum change

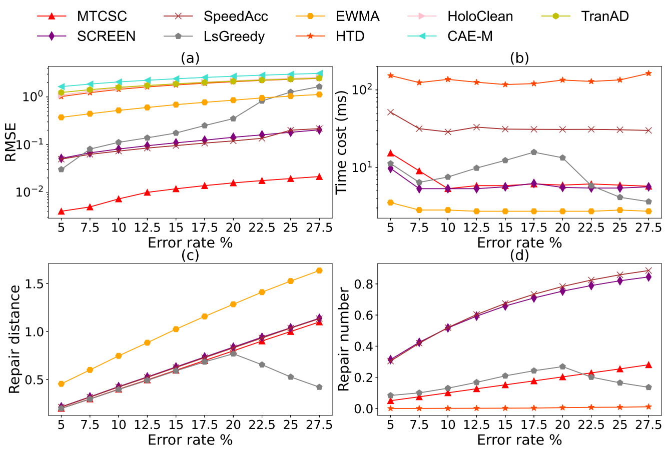

Figure 6: Varying error rate on Temperature (ILD)

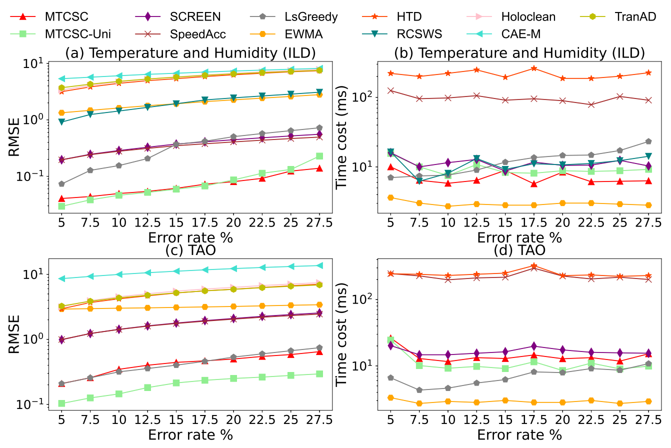

Figure 8: Varying error rate on (a-b) ILD (c-d) TAO (separate)

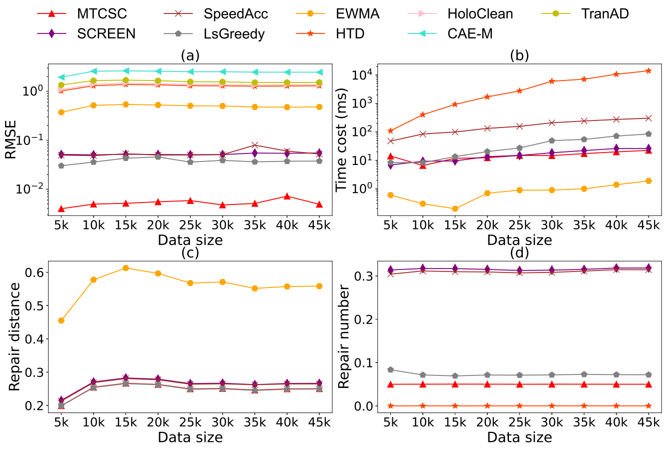

Figure 7: Varying data size on Temperature (ILD)

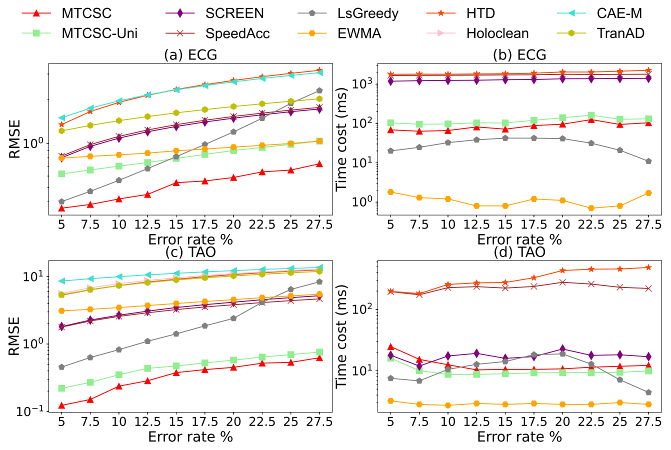

Figure 9: Varying error rate on (a-b) ECG (c-d) TAO (together)

principle, modify many more points than our proposed MTCSC, which is based on the minimum fix principle, resulting in an even larger repair distance. Moreover, the number of points modified by MTCSC is relatively similar to the number of manually injected errors. Considering the accuracy shown in Figure 6(a), the use of *minimum fix principle* makes sense.

5.2.3 Varying Data Size n. Figures 7(a) and (b) show that MTCSC also has good scalability in terms of data volume, achieves high accuracy and has a similar time cost as most fast methods with the exception of the EWMA. Figure 7(d) shows that MTCSC repairs about 5% of the points in each data size, which corresponds to the number of injected errors and indicates that MTCSC mainly cleans dirty points. It can be seen that both the machine learning-based HoloClean and the deep learning-based TranAD and CAE-M perform relatively poorly on univariate data.

5.3 Comparison on Multivariate Data

We perform experiments on multivariate time series with different error rates, data sizes and error patterns. Like other univariate methods, we also evaluate MTCSC-Uni by applying MTCSC in each dimension separately to analyze whether considering the entire dimensions as a whole has an effect.

5.3.1 Varying Error Rate e%. Figure 8(a) shows that our MTCSC and MTCSC-Uni achieve higher accuracy on ILD. Figures 9(a) and (b) show that MTCSC also has better accuracy with significantly less time compared to SCREEN and SpeedAcc on high-dimensional (32) ECG data. In Figure 8(c), the performance of LsGreedy on separate patterns on TAO is comparable to that of MTCSC, but MTCSC-Uni outperforms LsGreedy by far. In Figure 9(c), MTCSC and MTCSC-Uni lead in performance on TAO, with MTCSC slightly ahead of MTCSC-Uni. Figures 8(b) and (d), Figures9(b) and (d) show that MTCSC and MTCSC-Uni do not lose efficiency when performance is increased.

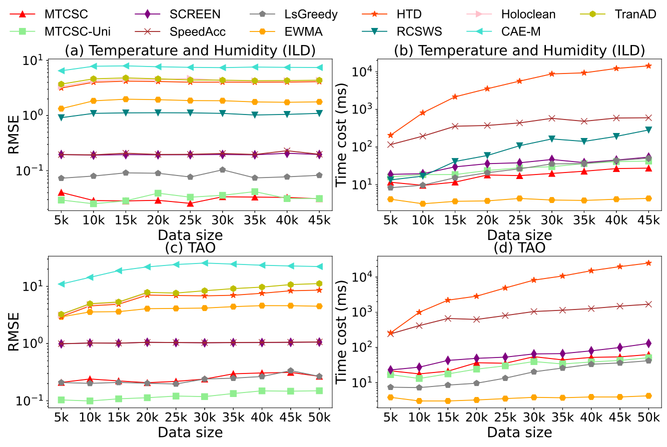

Figure 10: Varying data size on (a-b) ILD (c-d) TAO (separate)

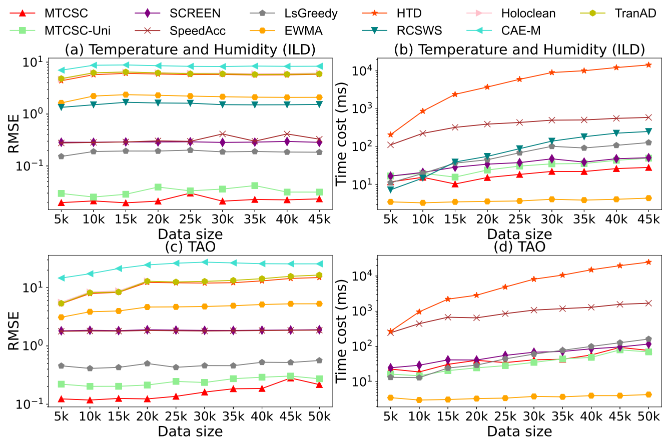

Figure 11: Varying data size on (a-b) ILD (c-d) TAO (together)

5.3.2 Varying Data Size n. Similarly, Figure 10(a) and Figure 11(a) show that MTCSC and MTCSC-Uni outperform others on ILD over different data sizes. In Figure 10(c), LsGreedy shows comparable performance to MTCSC, but is still worse than MTCSC-Uni. In Figure 11(c), the two proposed methods show slightly better performance than LsGreedy. Figures 10 and 11 confirm that MTCSC and MTCSC-Uni scale well with large datasets.

5.3.3 Varying Error Pattern. We inject two error patterns in multivariate data: "separate" and "together". (1) "Separate" means that we inject errors into each of the dimensions independently. For example, 2.5% of the points will have errors in one dimension, while another 2.5% will be affected in another dimension, when we inject 5% errors into two-dimensional data. (2) "Together" means that all dimensions of the randomly selected data points are affected simultaneously. This is to simulate the real case that a sensor can collect different types of data (e.g., temperature and humidity in ILD) at the same time, so that errors that occur due to physical or transmission failures should occur simultaneously at a given time.

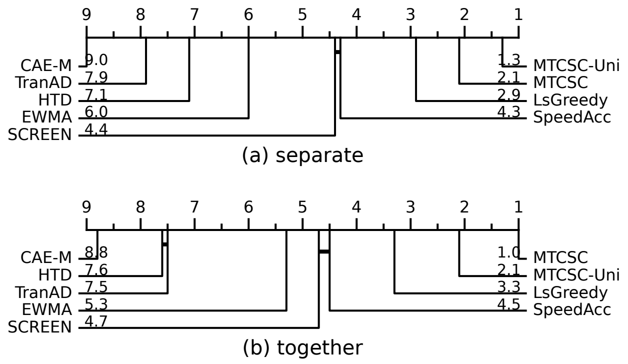

Figure 12: Critical difference diagram on Error Pattern

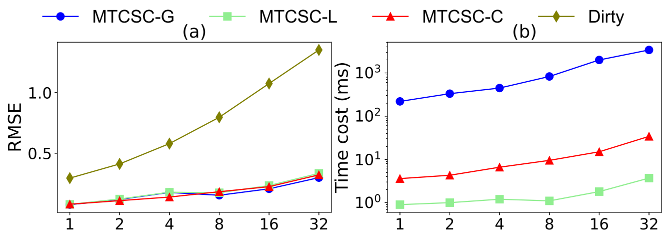

Figure 13: Varying dimension on ECG

Our MTCSC outperforms others (including MTCSC-Uni) under "together" pattern, as shown in Figures 9 and 11. Under "Separate" pattern, MTCSC performs similarly with the most competitive baseline LsGreedy, while MTCSC-Uni is a bit better. In contrast, both MTCSC and MTCSC-Uni are better on ILD. This is because as the number of dimensions increases, the error that occurs in a single dimension has little impact on the entire data, making it difficult for MTCSC to detect and repair.

Figure 12 shows the critical difference diagram [13] for the error pattern. Methods that are not connected by a bold line differ significantly in their average ranks. In real cases where errors occur individually, it is therefore recommended to use MTCSC-Uni. If errors can occur between multiple dimensions, as with GPS data, MTCSC is the best choice.

5.3.4 Varying Dimension D. We conduct experiments with different dimensions on ECG (32 dimensions in total). Figure 13(b) verifies the theoretical analysis that the time complexity of MTCSC-G, MTCSC-L and MTCSC-C are linear functions of the number of dimensions and shows that they can be executed in a reasonable time. Although it is difficult to determine the appropriate speed constraint as the dimension increases, our method can still achieve good results, as can be seen in Figure 13(a).

5.4 Comparison on Real Errors

We first compare the accuracy of error repair, and then validate the effectiveness of the adaptive method and explore its sensitivity.

5.4.1 GPS Trajectory with Human Walking. Table 4 shows the results on GPS dataset and the proposed MTCSC-C performs the

Table 4: GPS data with manually labeled ground truth

	RMSE	repair distance	repair number
Dirty	1.3553	-	-
MTCSC-G	0.4115	0.1134	163(1.52%)
MTCSC-L	2.1569	0.2241	286(2.66%)
MTCSC-C	**0.3386**	0.1256	185(1.72%)
MTCSC-Uni	0.4098	0.1178	160(1.49%)
RCSWS	1.2096	0.0571	179(1.66%)
SCREEN	0.9082	0.0925	284(2.64%)
SpeedAcc	0.9065	0.0928	286(2.66%)
LsGreedy	0.917	0.061	255(2.37%)
EWMA	2.0859	1.4236	10753(99.99%)
HTD	0.954	0.0224	41(0.38%)
Holoclean	1.1733	-	-
CAE-M	159.5	-	-
TranAD	35.98	-	-

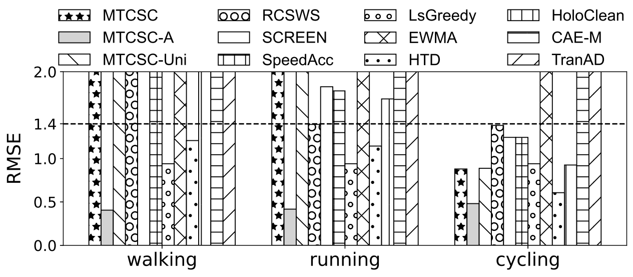

Figure 14: Adaptive Speed with Different Transportation

best. It is also worth noting that MTCSC-L is significantly worse than the others. This is due to the characteristics of the collected walking data. Walking through buildings or under bridges leads to consecutive errors, and these consecutive errors become the majority in our collection (the longest error sequence here contains 17 data points). MTCSC-L only considers the previous repair and the next incoming point and is probably misled by the occurrence of consecutive errors. In this scenario, we indeed observe the incapacity of almost all competitors. RCSWS suffers from oversimplified considerations regarding the data, while SCREEN and SpeedAcc are hampered by the separate consideration of speed in the dimensions. LsGreedy may not work optimally if only displacements of trajectory points occur without significant speed changes. HTD cannot recognize most errors and remains unchanged. It is possible that the learning-based methods cannot learn the pattern due to a lack of sufficient training data and achieve poor results.

5.4.2 Different Correlations. The superior performance of MTCSC-C compared to MTCSC-Uni in Table 4 shows that for data with correlations over Euclidean distance, the speed constraint over all dimensions together is the correct solution. Interestingly, the results on ILD, TAO and ECG (Figures 8 to 11) show that our MTCSC can still achieve good results as long as the speed is similarly scaled in different dimensions, even if there are no correlations between the dimensions over the Euclidean distance.

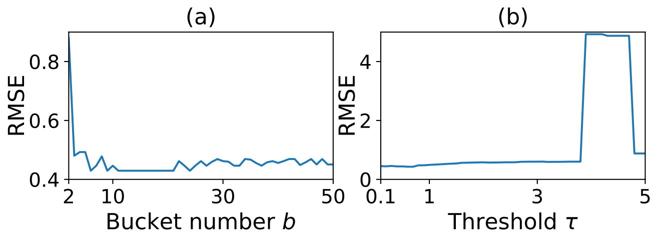

Figure 15: Sensitivity on bucket number b and threshold τ

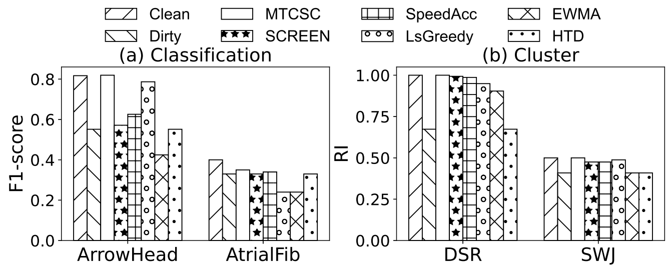

Figure 16: Classification and Cluster over clean, dirty and repaired data

5.4.3 Adaptive Speed with Different Transportation. We conduct experiments on another collected GPS dataset with three transportation modes: walking, running and cycling. We set the (initial) speed constraint corresponding to walking (1.6m/s), running (3.33m/s) and cycling (5.00m/s), respectively. The hyper-parameters of MTCSC-A are $b = 6$, $\tau = 0.75$, $w = 150$ and $\beta = 0.75$.

The dashed line in Figure 14 represents the RMSE of the dirty data. It is obvious that our MTCSC-A works optimally regardless of the initial setting. When the initial speed is set to walking or running, similar to EWMA, the constraint-based methods behave poorly because they fix too many correct points (the cycling part). Lsgreedy remains unaffected as it has nothing to do with the speed setting. HTD performs relatively well with extra labels for the sake of execution (which is indeed unfair).

We perform experiments over bucket number b and threshold τ (the other two are shown in Figure 4) to test the sensitivity. Figure 15(a) shows that MTCSC-A has a strong robustness to the bucket number b. For the threshold τ, the peak value observed in Figure 15(b) means that a relatively large threshold delays the detection time of speed changes and thus causes an excessive change in correct points. The subsequent sharp drop indicates that no more speed changes are detected if the threshold is even larger. Therefore, the setting of the threshold should also be careful.

5.5 Applications

We perform classification and clustering task to investigate the impact of different data cleaning methods on data mining applications. Specifically, we evaluate the performance of the original data as "Clean", introduce 10% random errors as "Dirty", and repair the data using different methods. All four datasets are naturally divided into training and testing sets, where we introduce random errors only in the training set and leave the testing set unchanged. For the multivariate time series AtrialFib and SWJ we inject errors with "together" pattern. The clustering task only uses the training sets. For classification, we use KNN [11] classifier, select the best K via a grid search and take the F1 score [2] as the evaluation measure. For clustering, we use K-means [26] and take the RI [29] as the evaluation measure.

As expected, Figure 16 shows that our MTCSC performs better than others and is similar to the results of "Clean". After cleaning, the distance between the time series is calculated more correctly.

5.6 Highlights and Limitations

We summarize the experimental highlights as follows: 1) Our proposal shows superior performance in cleaning time series compared to existing methods, with accuracy remaining stable even at high error rates (above 20%) and good scalability for large datasets; 2) The *minimum fix principle* can lead to smaller repair distances and increase repair accuracy compared to the minimum change principle as it suggests fewer changes to the originally correct points. In addition, tracking the trends of succeeding points can increase accuracy compared to border repairs; 3) Under the scenario that the sensor suffers some accidents, that errors occur in each dimension simultaneously (together), the proposed MTCSC achieves a lower RMSE than other competitors, especially the univariate cleaning methods, which follows the intuition that we should consider the multivariate case as a whole; 4) Once we know that errors occur individually, MTCSC-Uni is a better choice; 5) MTCSC-C can also perform an accurate cleaning of high-dimensional data in a reasonable time. 6) MTCSC-C can be effective with real GPS data with consecutive errors. 7) We are surprised that MTCSC can achieve satisfactory repair results even when there is no correlation in terms of Euclidean distance between dimensions, once the speed in each dimension is scaled to a uniform magnitude (the smaller the difference, the better the result); 8) MTCSC-A accurately detects speed changes and thus achieves better repair results, relieving users of the difficult decision of setting an appropriate speed limit; 9) Data cleaning promotes the accuracy of data science applications, and ours behave best. In short, we propose MTCSC-C, which uses only adjacent data points to identify the range of repair candidates, and MTCSC-A to dynamically determine the speed constraint to overcome the existing challenges 2 and 3 addressed in Section 1.2.

On the other hand, our work still has some limitations. 1) As mentioned in [33], the arrival rate (the number of data points in a period) may vary and the data points may be delayed due to network reasons, but in this work we only assume that all points arrive in the correct order; 2) Although we introduce MTCSC-A to enable adaptive speed constraint, choosing an appropriate window size is still an open problem; 3) The reason for our highlight in terms of correlation between data dimensions is not yet clear (challenge 1).

6 RELATED WORK

6.1 Traditional Cleaning

Constraint-based Cleaning. Constraint plays an important role in time series repairing. Holistic cleaning [10] is the initial method supporting speed constraints, but it is designed for relational data. [18] focuses on the difference between consecutive data points under sequential dependency, but it lacks precise expression of speed constraints. SCREEN[33] proposes an online cleaning method under speed constraints but can only handle univariate data. SpeedAcc [32] further considers the acceleration constraints in univariate time series. HTD [41] considers both the dimensional correlation and temporal correlation, but relies heavily on the difference between labeled truth and the observations. RCSWS [15] repairs the GPS data based on range constraints and sliding window statistics. Our MTCSC also takes advantage of data distribution and is applicable in multivariate time series cleaning.

Smoothing-based Cleaning. Moving average [7] is widely used to smooth time series data and make predictions. The simple moving average (SMA) calculates the unweighted mean of the last k data points and forecasts the next value. Alternatively, the weighted moving average (WMA) assigns different weights to data based on their positions. The exponentially weighted moving average (EWMA) [16] assigns exponentially decreasing weights over time. These techniques always modify a large portion of the data, which may change the data distributions in original data.

Statistical-based Cleaning. Statistical-based methods are also employed in time series cleaning. LsGreedy [38] establishes a probability distribution model of speed changes between adjacent points, which overcomes the problem that speed constraint cannot detect small errors. [20] presents an incremental clustering method where historical data is initially clustered, and the average values of the clusters are then used as the repaired result. STPM [28] learns detailed data patterns from historical data and applies them to clean current data.

6.2 Machine Learning-based Cleaning

HoloClean [30] is a weakly supervised learning system that builds a probabilistic model for cleaning. In order to cope with continuous data, we modify the quantization part and constraint parser part to promote the effectiveness on handling time series data with speed constraint. TranAD [35] is a prediction-based anomaly detection model that combines transformer-based encoder-decoder networks with adversarial training. CAE-M[39] is an auto-encoder based anomaly detection model that combines convolution with a memory network that includes Autoregressive model and Bidirectional LSTM with Attention.

7 CONCLUSION

In this paper, we study the data cleaning problem in multivariate time series. Existing methods based on the minimum change principle manage to minimize the repair distance, but lead to border repair. On the other hand, current constraint-based methods all focus on univariate time series. Even if the speed constraint is satisfied in each dimension, the constraint is still violated in multiple dimensions. To solve the above problems, we propose MTCSC, which is based on the minimum fix principle and applies the speed constraint in all dimensions together.

We first formalize the cleaning problem and propose MTCSC-G to obtain the global repair. Then, we propose MTCSC-L to support online cleaning. To balance the efficiency and effectiveness, we develop MTCSC-C by considering both data distribution and speed constraint. In addition, MTCSC-A is introduced to dynamically capture the speed constraint in real-world scenarios. Experiments on various datasets with different error rates, data sizes and error patterns show that our proposals have good repair performance, robustness and scalability. The reason why the speed constraint can be effective on datasets with weak/no correlations is interesting and remains an open problem.

REFERENCES

[1] Foto N. Afrati and Phokion G. Kolaitis. 2009. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT (ACM International Conference Proceeding Series, Vol. 361)*. ACM, 31–41.

[2] Charu C Aggarwal. 2016. Outlier analysis second edition.

[3] Fabrizio Angiulli and Fabio Fassetti. 2007. Detecting distance-based outliers in streams of data. In *CIKM*. ACM, 811–820.

[4] Anthony J. Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn J. Keogh. 2018. The UEA multivariate time series classification archive, 2018. *CoRR* abs/1811.00075 (2018). arXiv:1811.00075 http://arxiv.org/abs/1811.00075

[5] Ane Blázquez-García, Angel Conde, Usue Mori, and José Antonio Lozano. 2022. A Review on Outlier/Anomaly Detection in Time Series Data. *ACM Comput. Surv.* 54, 3 (2022), 56:1–56:33.

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

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

[8] Samuel Burer and Anureet Saxena. 2011. The MILP road to MIQCP. *Mixed integer nonlinear programming* (2011), 373–405.

[9] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. 2015. The UCR Time Series Classification Archive. www.cs.ucr.edu/~eamonn/time_series_data/.

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

[11] Thomas M. Cover and Peter E. Hart. 1967. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* 13, 1 (1967), 21–27.

[12] Tamraparni Dasu and Ji Meng Loh. 2012. Statistical Distortion: Consequences of Data Cleaning. *Proc. VLDB Endow.* 5, 11 (2012), 1674–1683.

[13] Janez Demsar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.* 7 (2006), 1–30. http://jmlr.org/papers/v7/demsar06a.html

[14] Guohui Ding, Yueyi Zhu, Chenyang Li, Jinwei Wang, Ru Wei, and Zhaoyu Liu. 2023. Time Series Data Cleaning Method Based on Optimized ELM Prediction Constraints. *J. Inf. Process. Syst.* 19, 2 (2023), 149–163.

[15] Chenglong Fang, Feng Wang, Bin Yao, and Jianqiu Xu. 2022. GPSClean: A Framework for Cleaning and Repairing GPS Data. *ACM Trans. Intell. Syst. Technol.* 13, 3 (2022), 40:1–40:22.

[16] Everette S. Gardner. 2006. Exponential smoothing: The state of the art—Part II. *International Journal of Forecasting* 22, 4 (2006), 637–666. https://doi.org/10.1016/j.ijforecast.2006.03.005

[17] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

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

[19] LLC Gurobi Optimization. 2024. Gurobi Optimization. https://www.gurobi.com/. Accessed: 2024-03-30.

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

[21] Rob J Hyndman and George Athanasopoulos. 2018. *Forecasting: principles and practice*. OTexts.

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

[23] H Zar Jerrold. 1999. Biostatistical analysis. *Biostatistical analysis* (1999).

[24] Solomon Kullback. 1997. *Information theory and statistics*. Courier Corporation.

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

[26] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[27] Samuel Madden. 2003. Intel Berkeley research lab data. https://db.csail.mit.edu/labdata/labdata.html. Accessed: 2024-04-10.

[28] Mostafa Milani, Zheng Zheng, and Fei Chiang. 2019. CurrentClean: Spatio-Temporal Cleaning of Stale Data. In *ICDE*. IEEE, 172–183.

[29] William M Rand. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association* 66, 336 (1971), 846–850.

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

[31] Craige Schensted. 1961. Longest increasing and decreasing subsequences. *Canadian Journal of mathematics* 13 (1961), 179–191.

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

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

[34] C Robert Taylor. 2019. Dynamic programming and the curses of dimensionality. In *Applications of dynamic programming to agricultural decision problems*. CRC Press, 1–10.

[35] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. 2022. TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data. *Proc. VLDB Endow.* 15, 6 (2022), 1201–1214.

[36] Haoyu Wang, Aoqian Zhang, Shaoxu Song, and Jianmin Wang. 2024. Streaming data cleaning based on speed change. *VLDB J.* 33, 1 (2024), 1–24.

[37] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. 2020. Ultrafast local outlier detection from a data stream with stationary region skipping. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1181–1191.

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

[39] Yuxin Zhang, Yiqiang Chen, Jindong Wang, and Zhiwen Pan. 2021. Unsupervised Deep Anomaly Detection for Multi-Sensor Time-Series Signals. *CoRR* abs/2107.12626 (2021).

[40] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. 2008. Learning transportation mode from raw gps data for geographic applications on the web. In *WWW*. ACM, 247–256.

[41] Jingjing Zhou, Xiaokang Yu, Jilin Zhang, Hanxiao Shi, Yuxin Mao, and Junfeng Yuan. 2022. A High-Dimensional Timing Data Cleaning Algorithm for Wireless Sensor Networks. *Ad Hoc Sens. Wirel. Networks* 53, 1-2 (2022), 141–164.