A PROOFS

A.1 Proof of Proposition 3.3

The sum of distances is easy to check in polynomial time, i.e., the problem is in NP. To prove the NP-Hardness, we show a reduction from the decision version of the Traveling Salesman Problem (TSP), one of Karp's 21 NP-Complete problems [26].

We first introduce the decision version of the TSP problem: Given an undirected weighted graph $G = (V, E)$ with n vertices and their pairwise distances, the problem is to decide whether there exists a path of total weight less than or equal to L, which visits each vertex exactly once. In addition, if all the vertices are in a plane (i.e., the distances satisfy the triangle inequality), existing studies also prove that the TSP problem is still NP-Complete under l_1 and l_2 distances [21, 24, 32].

Given an instance of the TSP problem in a plane, we then construct an instance of our Master Data Ordering Problem with 2 dimensions: For each vertice $v_i \in V$, we use $(v_i[1], v_i[2])$ to denote its coordinates, and then construct a tuple $y_i = (v_i[1], v_i[2]) \in Y$ corresponding to v_i. Therefore, the distances between vertices v_i, v_j are also equal to the distances between tuples y_i, y_j.

Following Problem 2, the decision version of our problem is to decide if there exists a master data order with the sum of the distances less than or equal to K. Let $K = L$, we prove that these two instances are equivalent. Note that the distances of two instances are equal, to find a path of total weight less than or equal to L in TSP problem, we then order the tuples following the order of the vertices in the path. Therefore, the path in TSP has total weight less than or equal to L, if and only if there exists an order of tuples with the sum of the distances less than or equal to K. To conclude, the NP-Completeness of the decision problem is proved.

A.2 Proof of Lemma 3.4

First, we consider the distance between x and y in the j-th dimension. When $Y_i^{min}[j] < x[j] < Y_i^{max}[j]$, $x[j]$ is covered by the range of $Y_i[j]$. Thereby, $\theta_j(x, Y_i) = 0$ in this case. When $x[j] < Y_i^{min}[j]$, $x[j]$ is out of the value range of the j-th dimension. Then the value closest to $x[j]$ is $Y_i^{min}[j]$, which gives the minimum distance $|x[j] - Y_i^{min}[j]|$. Similarly, we could get the distance bound of the j-th dimension when $x[j] > Y_i^{max}[j]$. In summary, we have

$$|x[j] - y[j]| \geq \theta_j(x, Y_i), \quad \forall y \in Y_i, \quad 0 \leq j \leq l$$

The distance $\delta(x, y)$ is thus bounded by $\theta(x, Y_i)$, for $y \in Y_i$.

$$\delta(x, y) = \sum_{j=1}^{l} |x[j] - y[j]| \geq \sum_{j=1}^{l} \theta_j(x, Y_i) = \theta(x, Y_i)$$

A.3 Proof of Proposition 3.5

Suppose that there exists a neighbor y^* of x in cluster Y_i, i.e., $\delta(x, y^*) \leq \delta_k$. On the other hand, according to the Lemma 3.4, $\forall y \in Y_i, \delta(x, y) \geq \theta(x, Y_i)$. It follows $\delta(x, y^*) \geq \theta(x, Y_i)$. We thus obtain $\theta(x, Y_i) \leq \delta_k$.

A.4 Proof of Proposition 3.6

The proposition can be proved similarly following the proof of Proposition 3.5.

A.5 Proof of Proposition 4.9

Note that the repair tuples in our algorithm are selected from the valid repair set V_i. Thereby, the repair x_i' of x_i exists if and only if V_i is not empty. We discuss two cases of window W_i.

If $W_i = \emptyset$, the smoothness constraint $\mathcal{S}$ of x_i is naturally satisfied. Therefore, we can always find a candidate from $knn(x_i)$ in the master data to repair x_i safely.

If $W_i \neq \emptyset$, then the candidate repair set C_i involves $knn(x_j')$, for $x_j' \in W_i$, i.e., $x_j' \in C_i$ as well. Due to the streaming computation, x_j is repaired before x_i. That is, x_j' in the window satisfies the smoothness constraint with each other, referring to the previous repair steps. Therefore, there always exists $x_j' \in W_i$ as the valid repair of x_i.

A.6 Proof of Proposition 4.12

For window size $\omega = 0$, the temporal smoothness of the time series is no longer considered. The window W_i of each tuple x_i is always empty and the candidate repair set is $C_i = knn(x_i)$. Since the repairs of tuples no longer affect with each other, we have $V_i = C_i$. The one with the minimum distance to x_i in V_i returned by Algorithm 2 is indeed the optimal solution.