Approximate Aggregate Ordered Dependency
AAOD
02360503

Project Submitter - Yontatan Azarzar

Supervisors - Dr. Youngmann Brit , Prof. Benny Kimelfeld

1. Introduction

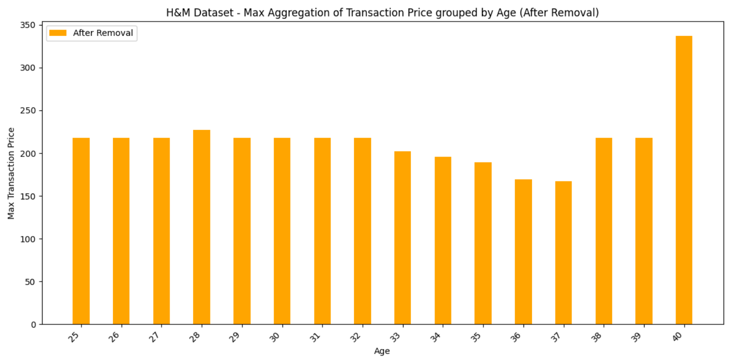
1.1 Monotonic Trend Analysis and Violation Vectors on Norms.

In many data analysis scenarios, we seek to identify and preserve monotonic trends within aggregated data. Specifically, when aggregating a certain attribute (e.g., price, salaries) grouped by another categorical variable (e.g., age, educational level), we expect the aggregated values to follow a monotonic pattern - either non-decreasing or non-increasing. Such monotonicity can reveal underlying consistent behaviors or trends in the data, which are often critical for business insights and decision making. Common aggregation functions used in monotonicity analysis include minimum, maximum, sum, median, and mean, each providing different insights into the distribution and trends of the grouped data.

However, real-world data rarely follows perfect monotonicity due to noise, errors, or outliers. To quantify deviations from this ideal monotonic trend, we measure violations - instances where the expected order is broken. Two common ways to define these violations are Consecutive Violation Vector (CVV) and All-Pairs Violation Vector (AVV). In our analysis, we focus on the violations between the aggregated group values (i.e., between columns after aggregation) rather than violations between individual tuples.

1.2 Consecutive Violation Vector (CVV)

Consecutive Violation Vector (CVV) focuses on local inconsistencies by examining only adjacent groups in the ordered sequence. It identifies violations between consecutive pairs of groups, capturing direct, neighboring deviations from monotonicity. This approach is computationally efficient and highlights small, localized errors. However, its drawback is that it sometimes allows small local violations that may accumulate and cause a reversal of the overall trend. We will now illustrate this concept with an example on a real dataset:



CVV-L$_\infty$ results on $\tau = 20$

removals: 41/1048575

This plot illustrates a key drawback of relying solely on the local, consecutive-pairs perspective used in CVV. When only consecutive aggregated violations are considered, the algorithm may allow a sequence of small, incremental decreases that cumulatively result in a significant trend reversal. In this case, even though each local drop is minor and falls within the allowed tolerance, their accumulation leads to a global pattern where the overall monotonicity is lost - with the sequence eventually dipping and then rising again sharply. This highlights that CVV, when applied purely locally, can fail to capture broader, non-monotonic trends in the data.

1.3 All-Pairs Violation Vector (AVV)

All-Pairs Violation Vector (AVV), on the other hand, considers violations between every possible pair of groups, not limited to neighbors. By capturing all deviations across the dataset, AVV provides a more comprehensive view of monotonicity violations. This approach is more robust in detecting subtle and long-range inconsistencies but is computationally more expensive due to the larger number of comparisons.

1.4 Norms for Aggregating Violations

In this project, we operate over general norms to aggregate these violations, with a focus on two specific norm types: L_1 norm and L_∞ norm. The L_1 norm aggregates violations by summing their absolute values, providing a measure of the total amount of monotonicity violation. The L_∞ norm focuses on the single largest violation, highlighting the worst-case deviation.

The advantage of the L_1 norm lies in its sensitivity to all violations, offering a balanced view of overall data quality. However, it may sometimes underestimate the impact of extreme outliers. Conversely, the L_∞ norm's strength is in emphasizing the maximum violation, making it suitable for scenarios where even a single large deviation is critical. Its drawback is that it ignores the cumulative effect of smaller violations, which can also be important, as discussed erlier.

Together, the choice between CVV and AVV, as well as between L_1 and L_∞ norms, allows for flexible and tailored monotonicity analysis, balancing computational cost and analytical depth depending on the application needs.

1.5 AOD's and connection to prevoius work

Aggregate Order Dependencies (AODs) are constraints that enforce a monotonic relationship between the aggregated values of groups within a dataset. Formally, an AOD specifies that, when tuples are grouped by a particular attribute, the aggregate (e.g., MAX, AVG, SUM) over another attribute must be non-decreasing (or non-increasing) according to the group order. AODs are particularly relevant in scenarios where aggregated data should exhibit a consistent trend, such as performance metrics over time, income levels by age, or quality indicators by production batch. For More formal defenitions and details, I deeply reccomend looking out for Sunit Agmon's paper on AOD's.
While enforcing AODs ensures strict monotonicity, the requirement is often too rigid for real-world data, where small fluctuations or noise may cause numerous violations. This strict monotonicity constraint often leads to the removal of many tuples, even though a faithful trend reflecting the true state of the data could be achieved by removing fewer records. Repairing such violations under the standard definition often entails removing a large number of tuples, even when the trend is visually and semantically preserved.

In this work, we extend the concept of AODs to **Approximate** Aggregate Order Dependencies (AAODs). The AAOD framework relaxes the strict ordering constraint by introducing a tolerance parameter (τ) that allows minor deviations from monotonicity. This tolerance is measured via a violation function, such as the Consecutive Violation Vector (CVV) or All-Pairs Violation Vector (AVV), and can be evaluated under different norms (e.g., L_1,L_∞). By doing so, we enable a principled trade-off between strictness and data retention, reducing the number of tuple deletions required while still maintaining an overall monotonic pattern in the aggregated data.

2. Formal Notations & Definitions

r_0 : a relation over schema S (i.e., a dataset).
A : an attribute over an ordered domain (e.g., age group).
We ssume $A = \{a_1, a_2 ... a_n\}$ is ordered increasingly ($a_i < a_j \iff i < j$).
B : a numeric attribute (e.g., salary)
$\alpha_r(B|A = a)$: an aggregate over B, grouped by A. α can be mean , sum etc.

$\tau \in [0, \infty)$: tolerance for approximate monotonicity.
To quantify how badly the relation r violates the monotonicity (under CVV or AVV), we use norms.
Let $VV(r)$ denote the violation vector of relation r , under scope $VV \in \{CVV, AVV\}$.

2.1 Consecutive Violation Vector (CVV)

CVV captures only local violations , i.e., between adjacent pairs in the ordering of A.

$$CVV_i = \max(\alpha_r(B|A = a_i) - \alpha_r(B|A = a_{i+1}) \ , \ 0) \ .$$

Each entry measures how much the aggreate drops from a_i to the adjacent, so we are looking only on immediate neighbors. vector size = n-1.
Therefore $CVV(r) \in \mathbb{R}_{\geqslant 0}^{n-1}$.

2.2 All-Pair Violation Vector (AVV)

AVV captures all violations of monotonicity of all ordered pairs (a_i, a_j) with $i < j$.

$$AVV_{i,j} = \max(\alpha_r(B|A = a_i) - \alpha_r(B|A = a_j) \ , \ 0) \ \ \forall i<j \ .$$

Each entry measures how much the aggreate drops from a_i to a_j.
The AVV is a vector of size $\binom{n}{2}$, for all unordered pairs. Therefore $AVV(r) \in \mathbb{R}_{\geqslant 0}^{\binom{n}{2}}$.

3. The formal AAOD problem

An *approximate aggregate order dependency* (AAOD) is an expression of the form:

$$A \overset{\tau,\, VV\,,\, \|\cdot\|}{\nearrow} \alpha(B)$$

where A,B are attributes in Att(S), $VV \in \{CVV, AVV\}$, norm $\|\cdot\|$ and tolerance $\tau \geqslant 0$.
We refer to A as the grouping attribute and to B as the aggregate attribute.

We say that a relation r satisfies the AAOD $A \overset{\tau, VV\,,\, \|\cdot\|}{\nearrow} \alpha(B)$, denoted $r \vDash A \overset{\tau,\, VV\,,\, \|\cdot\|}{\nearrow} \alpha(B)$ if the following hold:

$$\|VV(r)\| \leqslant \tau$$

3.1 Motivation Examples

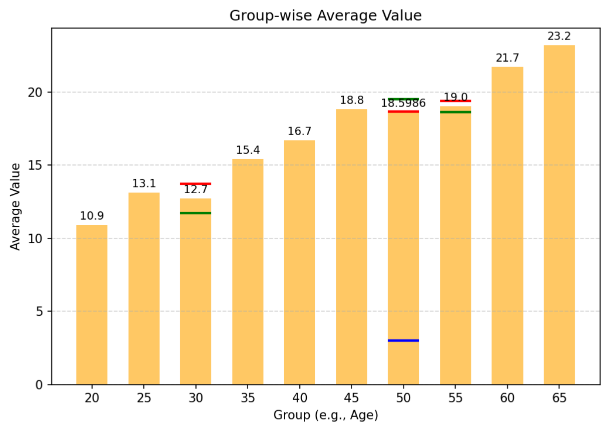
3.1.1 importance of using τ

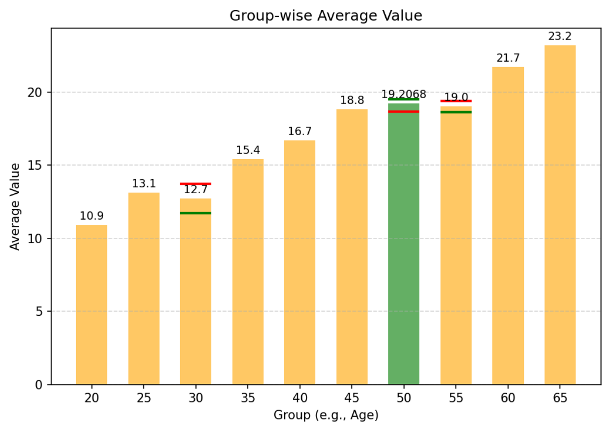
In the following scenerio , we will see that even one outlier can cause a massive removals from the dataset , when using "pure" AOD ($\tau = 0$).
The given dataset is cheacked over VV=CVV and $\|\cdot\| = L_1$ with $\tau = 0.6$.

The initial result of the norm (L1) is 1.2 (0.4+0.8). Here , we aim for $\tau = 0.6$ which means we decrease the total "penalty" by 50%.
To do so , we need to omit only 1 tuple (outlier). ($2.5 \cdot 10^{-5}$% of the data)

By contrast, enforcing the stricter condition where $\tau = 0$ would require removing 4001 times more tuples than before! This demonstrates the motivation for introducing epsilon - it enables us to capture nearly monotonic behavior while avoiding excessive data elimination.

Age	#tuples	Salary
20	4000	10.9
25	4000	13.1
30	2000	13.7
30	2000	11.7
35	4000	15.4
40	4000	16.7
45	4000	18.8
50	1	3 (outlier)
50	2575	19.5
50	1424	18.65
55	2000	19.35
55	2000	18.65
60	4000	21.7
65	4000	23.2





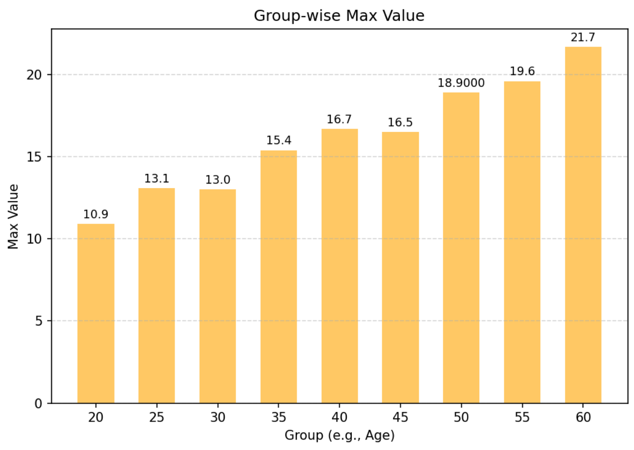
1

tuple of age 50 (the outlier with vakue=3)
2000 tuples of age 55 (those with value=18.6)
2000 tuples of age 30 (those with value=11.7)

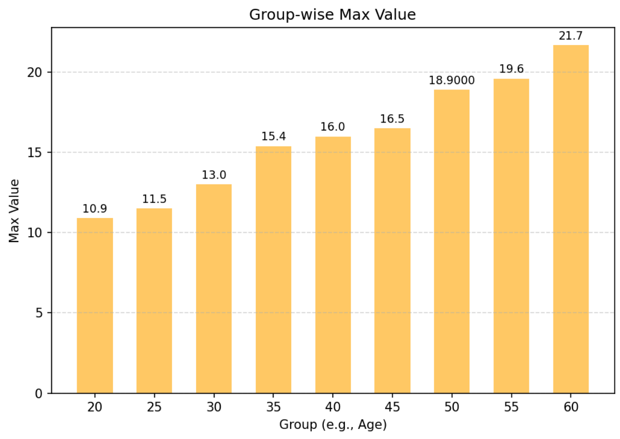
3.1.2 importance of using τ for CVV and $||\cdot|| = L_\infty$ agg = max

Age	#tuples	Salary
20	2000	10.9
25	1000	13.1
25	1000	11.5
30	2000	13.0
35	2000	15.4
40	1000	16.7
40	1000	16.0
45	2000	16.5
50	2000	18.9
55	2000	19.6
60	2000	21.7

$\tau = 0 \rightarrow$ 2000 tuples to remove



$\tau = 0.05 \rightarrow$ 1000 tuples to remove



1 tuple of age 50 (the outlier with vakue=3)
2000 tuples of age 55 (those with value=18.6)
2000 tuples of age 30 (those with value=11.7)

In the above chart, we observe the maximum value of each age group (e.g., 20, 25, ..., 60), where each group consists of 1000 individual records. The general trend of the max values appears to be increasing - which aligns with our intuition about age and certain attributes like income or experience.

However, under the CVV (Consecutive Violation Vector) framework, even a single drop between two consecutive max values is considered a violation of the aggregate order dependency (AOD),
specifically: $||\mathrm{VV(r)}|| = \ ||(0, 0, ..0.1, 0.2, 0, 0..)||_\infty = 0.2 > \tau$ for $\tau = 0.1$

In this chart, although the trend is broadly increasing, we observe a slight drop between: Group 25 (13.1) and Group 30 (13.0), Group 40 (16.7) and Group 45 (16.5)
These are very minor drops (e.g., 0.1 or 0.2), and visually, the sequence still looks almost perfectly increasing.

For the regular AOD constaraint (when $\tau = 0$), these tiny differences trigger violations, and since the aggregation is MAX, repairing such violations requires removing all tuples in the violating group that are greater than the allowed threshold — which in this case may mean hundreds of tuples per group.

This results in excessive data deletion due to minor noise, which is undesirable and unnecessary in most practical applications.

4. Algorithms to solve AAOD problem

In this section , we will present 4 algorithms to solve the AAOD problem, all based on dynamic programming for efficiency. The algorithms find minimal set to be removed from the dataset , so that the AAOD definition holds under the choosed parameters $\tau, \|\cdot\|, VV$. In all cases - "AggPack" is used at to compute the maximal subset of a group r_i whose aggregation equals exactly to x.

4.1 CVV & L_∞

The first algorithm checks for consecutive violations under L_∞ norm.
In this setting, we are very close to tha classic AOD algorithm - we allow a column i to have a lower aggregated value, as long as the difference with column i-1 is not greater than τ.
This approach ensures an <u>exact computation</u> of the minimal deletions needed to keep the constraint of CVV under L_∞ and a threshold τ.

```
input: Relation r , AAOD $\delta$
output: Maximal subset $r' \subseteq r$ such that $r' \models \delta$
$H_0[0] \leftarrow \emptyset$
for $i \in \{1,..,m\}$ do
    for $x \in V_\alpha(r_i)$ do
        $F_i[x] \leftarrow AggPack < \alpha > (r_i, x)$
    for $x \in F_i$ do
        $H_i[x] \leftarrow H^{\leqslant}_{i-1}[x+\tau] \cup F_i[x]$
    for $x \in H_{i-1} \setminus F_i$
        $H_i[x] \leftarrow H_{i-1}[x]$
return $H^{\leqslant}_m[\max(V_\alpha(r))]$
```

<u>**Notations**</u>

- $\mathbf{V_\alpha(r_i)}$ - vector of all possible aggregations on the any $r_i' \subseteq r_i$, under aggregation α (sum ,avg, count , etc.) .
- $\mathbf{AggPack < \alpha > (r_i, x)}$ - finds the acutal group for x in $V_\alpha(r_i)$.
- $\mathbf{H_i[x]}$ - best solution (minimal removals) of columns 1...i such that the AAOD holds and the aggregation of the i'th column is exactly x.

4.2 CVV & L_1

The second algorithm also checks for consecutive violations, this time under L_1 norm. In this approach, we measure monotonicity violations only between consecutive groups and sum their absolute magnitudes, yielding the L1 norm of the violation vector.

The dynamic programing method here , uses a "budget" that symbolize the overall violation that was already "in-use"in the solution to that point.

In each step , we save for each budget and aggregation value of the last column , what was the best solution (size). We ensure that the cumulative violation across all consecutive pairs remains within the threshold τ.

This method provides an exact minimal-deletion solution for controlling total local deviation from monotonicity.

```
input: Relation r , AAOD $\delta$
output: Maximal subset $r' \subseteq r$ such that $r' \models \delta$
$H_0[0, b] \leftarrow \emptyset$ for any $t \in \{0,\ldots,\tau\}$
$H_i[x, 0] \leftarrow \emptyset$ for any $x \in V_\alpha(r_i)$
for $i \in \{1,..,m\}$ do
    for $x \in V_\alpha(r_i)$ do
        $F_i[x] \leftarrow AggPack < \alpha > (r_i, x)$
    for $x \in F_i$ $b \in \{0,..,\tau\}$ do
        $H_1[x, b] \leftarrow \underset{b' \in \{0,..,b\}}{\text{argmax}}(H_{i-1}[x + b', b - b'] \cup F_i[x])$
        $H_2[x, b] \leftarrow \underset{x' \in \{0,..,x\}}{\text{argmax}}(H_{i-1}[x', b] \cup F_i[x])$
        $H_i[x, b] \leftarrow \text{argmax}\{H_1[x, b], H_2[x, b]\}$
    for $x \in H_{i-1} \setminus F_i$
        $H_i[x, b] \leftarrow H_{i-1}[x, b]$
return $\underset{x \in \{0..V_\alpha(r)\},\, b \in \{0..\tau\}}{\text{argmax}} (H_m(x, b))$
```

Notations:

- $H_i[x, b]$ - the best solution for the first i columns , such that (AVV & L_1) holds under τ with total violation of b and so that $\alpha(r_i) = x$. $H_i[x, b]$ is the best out $H_1[x, b]$, $H_2[x, b]$, in terms of sultion size.
- $H_1[x, b]$ - the best solution so that the (i-1)'th column has higher agg value than the i'th col, and the (i-1)'th agg value causes new violation of exactly b' .
- $H_2[x, b]$ - the best solution so that the i'th column has higher agg value than the (i-1)'th agg value , and the first equals to x. In this case , the is no additional violation

added to the budget.

4.3 AVV & L_∞

The third algorithm is a dynamic programing implementation for the constraint of AVV under L_∞ norm.
Here , we take different approach for "x" as the dynamic programing parameter. In this setting, "x" tracks the maximum aggregation on <u>all columns</u> from 1...i.
To find exact solution ,we consider two cases at each step - whether the maximum agregation originates from the current group or from previous groups - and update the optimal subset accordingly. For the first case , there are no new violations with any of the previous col's because the new aggregated value of i is greater than any of 1...i-1 cols. In the second case we know that there will be a violation of the new aggregated value of i , with the maximal agg value from 1...i-1 (because that is not the first case), and mathematically the largest new violation to consider is the the one created between the i'th col and the maximal one. That also gives motivation from the change in the "x" character in the DP.
This approach ensures an exact computation of the minimal deletions needed to keep the maximum violation under the given threshold τ.

```
input: Relation r , AAOD $\delta$
output: Maximal subset $r' \subseteq r$ such that $r' \vDash \delta$
$M_0[0] \leftarrow \varnothing$
for $i \in \{1,..,m\}$ do
    for $x \in V_\alpha(r_i)$ do
        $F_i[x] \leftarrow AggPack < \alpha > (r_i, x)$
    for $x \in M_{i-1} \cup F_i$ do
        $M_1 \leftarrow M_{i-1}\left[\underset{y \in H_i \wedge y \leqslant x}{\operatorname{argmax}} |M_i[y]|\right] \cup F_i[x]$
        $M_2 \leftarrow M_{i-1}[x] \cup F_i\left[\underset{x-\tau \leqslant y < x}{\operatorname{argmax}} F[y]\right]$
        $M_i[x] \leftarrow \max(M_1, M_2)$
return $\underset{x \in \{0..V_\alpha(r)\}}{\operatorname{argmax}} (M_m(x))$
```

<u>**Notations:**</u>

- $M_i[x]$ - the best solution for the first i columns , such that (AVV & L_∞) holds under τ and that the solution hold the following : $\max_{j \leqslant i} \alpha(r_j) = x$. $M_i[x]$ gets its value from the best out of M_1, M_2.
- $M_i^{\leqslant}[x]$ - $M_i\left[\underset{y \in H_i \wedge y \leqslant x}{\operatorname{argmax}} |M_i[y]|\right]$

- M_1 - the i'th column has the maximal aggregated value (i.e, $\forall i' < i : \alpha(r_{i'}) < \alpha(r_i)$)
- M_2 - the maximal aggregated value in the first (i-1) columns and $x \geqslant \alpha(r_i) \geqslant x - \tau$.

4.4 AVV & L_1

The fourth and last algorithm , aims to solve the AAOD constraint under AVV-L_1. We emphasize that this is not an exact solution as discussed before, but a heuristic one. The reason for that derives from the "short memory" of DP to only the last iteration. Unlike the third algo' (AVV-L_∞) , here it not sufficient to have a look only on previous step , because in some cases , an optimal solution will have a lot of removals in a "midldle" column (not the largest grouped value) as a good tradeoff for least amount of removals in the future. That can not be captured in our algorithm because it does not memorize more than one step.

The idea in this DP approach is similar to $CVV - L_1$. We now follow a budget that hold all-pairs of violations in the current solution, and we demand that the new col adds a new violation so that the previous violation + new one (created by the i'th cols) matches exactly to the budget "b" parameter.

```
                **input :** Relation r , AAOD $\delta$
          **output :** Maximal subset $r' \subseteq r$ such that $r' \vDash \delta$
              $B_0[0,b] \leftarrow \emptyset$ for any $t \in \{0,\ldots,\tau\}$
              $B_i[x,0] \leftarrow \emptyset$ for any $x \in V_\alpha(r_i)$
              for $i \in \{1,..,m\}$ do
                for $x \in V_\alpha(r_i)$ do
                  $F_i[x] \leftarrow AggPack < \alpha > (r_i, x)$
                for $x \in F_i \cup B_{i-1}$ , $b \in \{0,..,\tau\}$ do
                  $temp1[x,b] \leftarrow \underset{x' \in \{0,..,x\}}{argmax}(B_{i-1}[x',b] \cup F_i[x])$
                  $temp2[x,b] \leftarrow \underset{\substack{x' \in \{x-(b-b'),..,x\} \\ b' \in \{0,\ldots,b\} \\ b'+penalty=b}}{argmax} (B_{i-1}[x,b'] \cup F_i[x'])$
                  $B_i[x,b] \leftarrow max\,(B_1, B_2)$
              return $B_m^{\leqslant}[max(V_\alpha(r)), \tau]$
```

Notations:

- $B_i[x]$ - the best solution for the first i columns , such that (AVV & L_1) holds under τ and that the solution hold the following : $\max_{j \leqslant i} \alpha(r_j) = x$.
- $B_i[x]$ gets its value from the best out of B_1, B_2.
- $B_i^{\leqslant}[x]$ - $M_i\left[\underset{y \in H_i \wedge y \leqslant x}{argmax} |B_i[y]|\right]$
- B_1 - the i'th column has the maximal aggregated value (i.e, $\forall i' < i: \alpha(r_{i'}) < \alpha(r_i)$)
- B_2 - the maximal aggregated value in the first (i-1) columns and $x \geqslant \alpha(r_i) \geqslant x - \tau$.

- penalty - the additional violation caused by adding the i'th column.

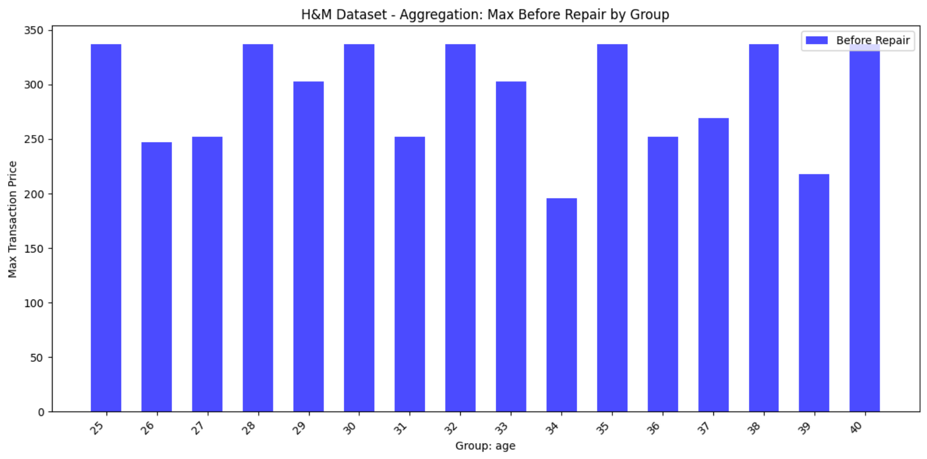
5. Experimental Analysis on a Real-World Dataset

In this chapter, we present a comprehensive analysis of the experimental results obtained from applying various AAOD methods on database datasets. Our focus lies on comparing the performance of algorithms under different norm constraints, specifically the L_1 and L_∞ norms, combined with consecutive penalty function (CVV) and all-pairs penalties (AVV). Due to computational reasons , we examined the max as the aggregation in all experiments. Through these experiments, we demonstrate how the careful selection of the parameter tau (τ) influences the quality and interpretability of the results. In particular, we compare the effects of using a permissive tau value against the stricter, more demanding requirement of tau equal to zero, highlighting the trade-offs between data removal rates and approximation accuracy. This analysis aims to provide insight into how tau tuning can
l ead to effective detection of monotonicity violations while minimizing unnecessary data exclusions.

The dataset used in our experiments is sourced from H&M and includes rich transactional data. In our analysis, we group the data by the column "age" and perform aggregation on the column "price", enabling us to examine meaningful patterns and detect outliers effectively.

5.1 Performance Results for Tau = 0 on the Dataset

For a tau value of 0, the algorithm achieved a removal of 99 records out of 1048575, demonstrating a strict enforcement of the monotonicity constraints. This approach, while precise, results in a higher number of exclusions, which may impact data retention. The following graph illustrates the aggregation of the dataset before and after the removal process, based on the maximum aggregation metric.

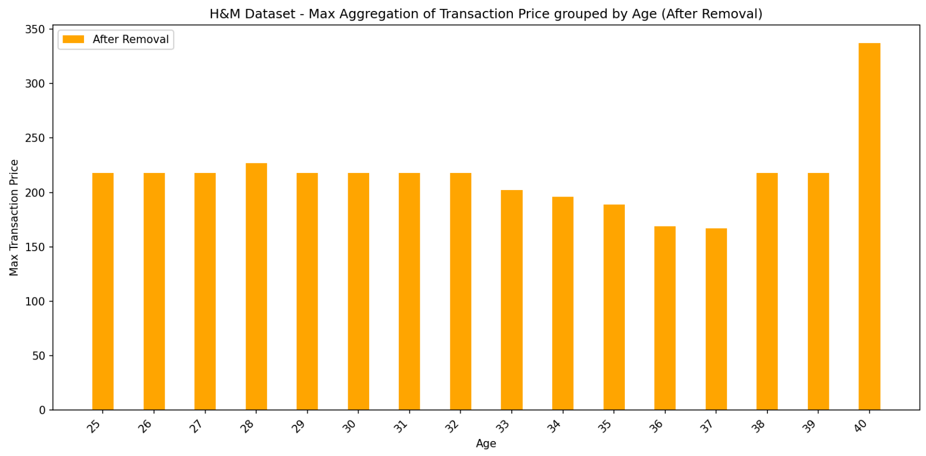


5.2 Case Study 2: Results for $\tau = 20$ (CVV L_∞ Penalty)

Using the CVV L_∞ penalty with a tau value of 20, we removed 41 records from the dataset. Prior to removal, the maximum penalty between consecutive groups was 119, where the average consecutive violation (mean max difference between groups) is about 66.9.

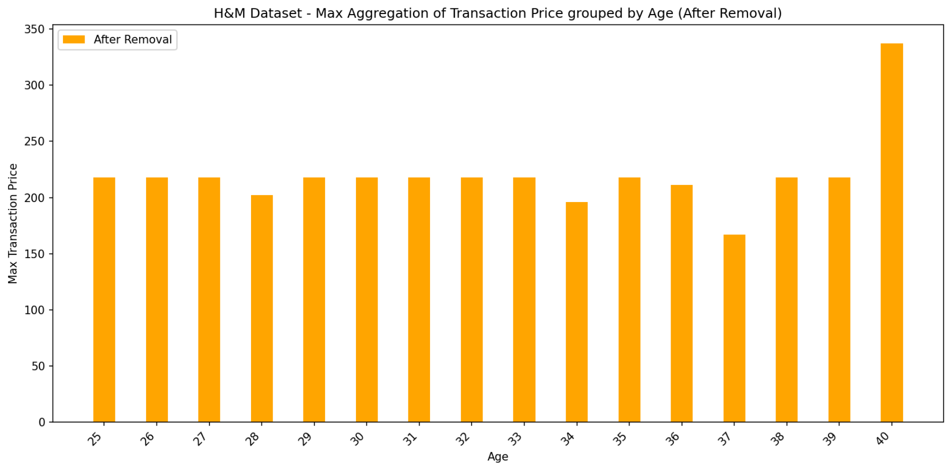
The tau value of 20 represents approximately 16.8% of the maximum consecutive violation penalty and about 29.9% of the average consecutive violation.

This demonstrates that by allowing a controlled penalty level relative to the dataset's natural variation, we can effectively balance data quality improvement with minimal data loss.



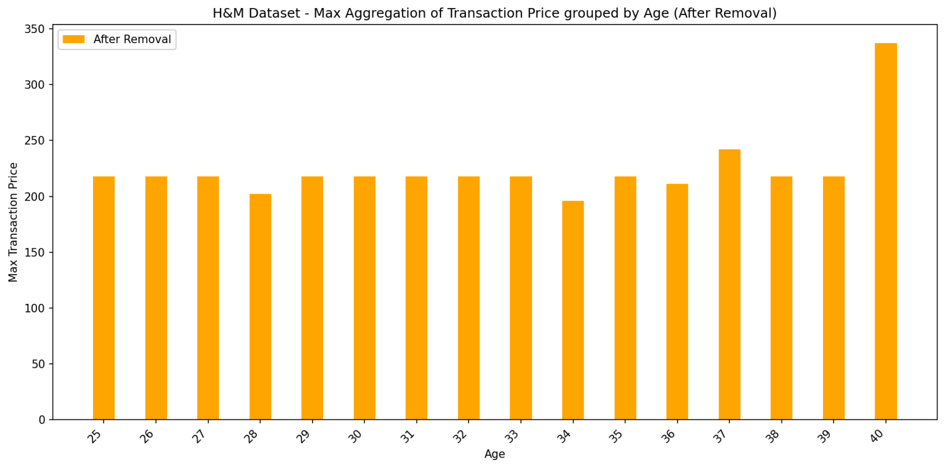
5.3 Case Study 3: Results for $\tau = 150$ (CVV L1 Penalty)

For a τ value of zero, we were able to achieve a solution that removes 99 records from the dataset. In contrast, using the CVV L1 approach, we reached a comparable accuracy with only 35 removed records, less than half the number at a tau value of 150.
The strong similarity between the resulting graphs highlights that even with a more permissive tau, it is possible to extract meaningful information about the monotonicity of the data. This approach significantly improves the efficiency of AAOD by reducing the number of removals required to achieve a desired level of precision, thereby enhancing data quality with minimal data loss.



5.4 Case Study 4: Results for $\tau = 30$ (AVV L_∞ Penalty)

In this experiment, the AVV L_∞ penalty with $\tau = 30$ was applied to the H&M dataset, using the maximum transaction price aggregated by age. The results show that only 34 tuples out of over 1,048,575 records had to be removed to satisfy the relaxed monotonicity constraint - a third of the $\tau = 0$ case. This extremely low removal rate demonstrates the strength of the AVV approach: by considering violations across all pairs of groups (rather than just consecutive ones), the method preserves nearly all the data while still enforcing a globally consistent ordering within the allowed deviation threshold. The use of the L_∞ norm ensures that no single violation exceeds τ, making the result robust to large local deviations and have a very good computional balance while allowing flexibility in smaller fluctuations.



5.5 Case Study 5: Results for $\tau = 50$ (AVV L1 Penalty)

In this experiment, the AVV L_1 penalty with $\tau = 50$ was applied to the H&M dataset, using the maximum transaction price aggregated by age. The total sum of penalties across all pairs of groups before repair was 554, and the removal using $\tau = 50$ (which is about 9% of total violations) resulted in removing 79 tuples (compared to 99 when $\tau = 0$). This result highlights the efficiency of the AVV L_1 approach, which minimizes the total accumulated violations , even with very small τ value, and preserves good the behavior of the plot when the stricter constraint is applied. The method balances local and global consistency while preserving the vast majority of the dataset (over 99.99% of records retained).

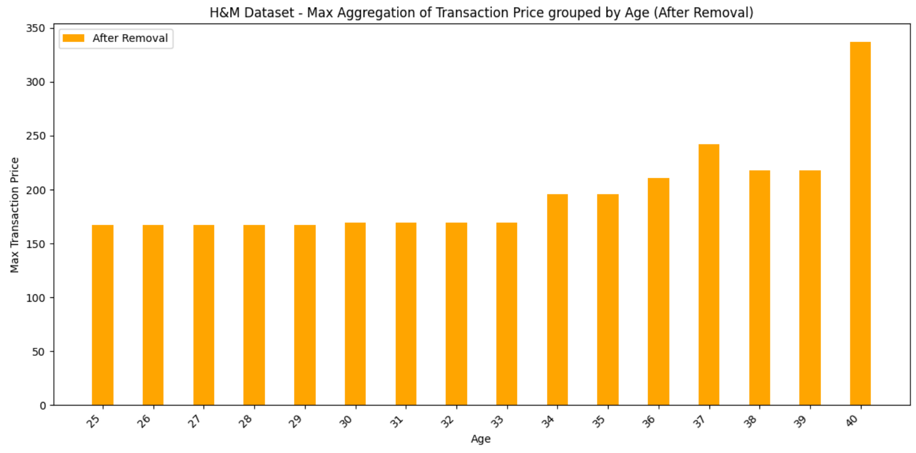


Figure 1: tau 50 AVVL1
removals 79
out of 1048575

6. Summary and Futrue work suggestions

6.1 Summary

in this work, we focused on Approximate Aggregate Order Dependency (AAOD) analysis as generalization of the classic AOD constraint (actually , AOD is a special case of AAOD with $\tau = 0$).
We discussed the motivation for extending our approach to AAOD (Approximate Aggregate Order Dependency), highlighting its importance for allowing controlled relaxations of strict constraints in large datasets. We first examined artificial but important examples illustrating the importance of non-zero τ, and analyzed the advantages and disadvantages of using different types of violation measures (consecutive/all-pairs). In particular, we compared violation based on CVV (Consecutive Violation Vector), which focuses on nearby pairs, with violation based on All Pairs (AVV), which considers all possible pairwise violations. This comparison helped clarify the trade-offs in precision and computational complexity when selecting a violation measure.

Using the CVV and AVV as the violation measurements , we implemented four algorithms that operate over CVV and AVV with the L_1 and L_∞ norms. Our experiments showed that the first three algorithms (CVV-L_1 , CVV-L_∞ , AVV-L_∞) produce exact solutions using only the previous step of the dynamic programing, and we demonstrated how it is possible to reduce the number of removals by relaxing the strict definition of violations and allowing some violations to remain.

The fourth algorithm, AVV-L_1, currently works as a kind of exact heuristic. While it produces reasonable results, it is not guaranteed to produce exact solution as a tradeoff of running in polynomial time.

6.2 Futrue Work Suggestions

For future research, there are several directions to explore:
AVV-L_1 Algorithm Improvement: Investigate whether a polynomial-time algorithm can be developed that solves the AVV-L_1 problem exactly, rather than relying on heuristic approaches.

Scalability and Efficiency: Study ways to improve or optimize some of the computational efficiency of all 4 algorithms, especially for large-scale datasets, possibly through pruning or approximation techniques.

The Cumulative Decrease Problem: Make improvements to the CVV algorithms so that they will better tolerate the cumulative decreases discussed in the begining of the work , for example by bounding the number of columns where violations are allowed or limiting the number of columns that can reverse the trend.