Properties of Inconsistency Measures for Databases

Samantha Berg
A20456450
sberg1@hawk.iit.edu

Hernan Razo
A20436834
hrazo@hawk.iit.edu

Christopher Anthony
A20496584
canthony1@hawk.iit.edu

Introduction

Livshitz et al. discuss how to quantify the inconsistencies of a database that violates integrity constraints. It is crucial to measure these inconsistencies to maintain proper function of typical database needs such as progress indication and action prioritization in cleaning systems. Specifically, progress indication has obvious applications in Human-Computer Interaction. This can be seen with creating accurate progress bar animations. Also, progress indication can be used to create recommendations for repairing data.

But to choose the correct inconsistency measure, we need to identify the needed properties in the application and understand which ones are least expected in practice.

Livshitz et al. try to answer this question by reviewing previously established measures from both knowledge representation (KR) and database communities. However, the authors of this paper devise properties in regards to operational aspects of repair systems. They analyze the computational complexity of these measures.

Specifically, they define four properties of inconsistency measures in the context of repair systems:

1. **Positivity** - A measure is strictly positive if and only if the database is inconsistent.
2. **Monotonicity** - Inconsistency cannot be reduced if the constraints get stricter
3. **Bounded Continuity** - A single operation can have a limited relative impact on inconsistency
4. **Progression** - We can always find an operation that reduces inconsistency

These properties are used to analyze a few inconsistency measures adapted from the KR and Logic literature. It is shown that most of these measures violate at least one of the previously listed properties. The inconsistency measures they focus on are:

1. $\mathbf{I'_{MC}}$ - The set of all self-inconsistencies in a database.
2. $\mathbf{I_d}$ - Drastic inconsistency value. It is the indicator function of inconsistency.
3. $\mathbf{I_{MI}}$ - MI Shapely Inconsistency. It is the cardinality of the set.
4. $\mathbf{I_P}$ - Count of facts that belong to the minimal inconsistent subset (problematic facts)
5. $\mathbf{I_{MC}}$ - The set of all maximal consistent subsets of a database
6. consistencies in a database
7. $\mathbf{I_R}$ - The minimal cost of a sequence of operations that repairs the database.

Also, they propose a new measure for satisfying all criteria. This new measure offers a combination of rationality and tractability. They go on and prove that this new measure satisfies all the properties stated earlier.

Satisfaction of Properties

The above stated inconsistency measures can satisfy the four properties or violate them. Livshitz et al. figure this out by using a case composed of a system of functional dependencies or denial constraints and a subset system as a repair system where only tuple deletions are allowed.

The authors then show proofs that show that only the inconsistency measures that are rational can be proven to be intractable. They also show that almost all inconsistency measures cannot satisfy at least one of the four properties. Their

findings can be summarized in the following table:

	Pos.	Mono.	B. Cont.	Prog.	PTime
$\mathcal{I}_d$	✓/✓	✓/✓	✗/✗	✗/✗	✓/✓
$\mathcal{I}_{MI}$	✓/✓	✓/✗	✗/✗	✓/✓	✓/✓
$\mathcal{I}_P$	✓/✓	✓/✗	✗/✗	✓/✓	✓/✓
$\mathcal{I}_{MC}$	✓/✗	✗/✗	✓/✓	✗/✗	✗/✗
$\mathcal{I}'_{MC}$	✓/✓	✗/✗	✗/✗	✗/✗	✗/✗
$\mathcal{I}_{\mathcal{R}}$	✓/✓	✓/✓	✓/✓	✓/✓	✗/✗
$\mathcal{I}^{lin}_{\mathcal{R}}$	✓/✓	✓/✓	✓/✓	✓/✓	✓/✓

There is an exception with I_R. At first, this seems promising, but further analysis in the next section shows that this measure is intractable in most cases and therefore, still not ideal.

Computational Complexity

Measuring inconsistencies and using repair systems obviously comes at a cost. Also, in practice, databases can get quite large. This adds to the need of having the most efficient way of measuring inconsistency for the application at hand.

To measure the computational complexity of each scenario, the authors focused on the class of denial constraints and the special class of functional dependencies. Additionally, the authors focused on data complexity, which means that they had a fixed set of integrity constraints

For I_d, the computational complexity relies on testing consistency. This can be done in polynomial time.

For both I_{MI} and I_P, these can be computed in polynomial time. This is possible by enumerating all the subsets in a database of a bounded size.

For I_{MC} and I'_{MC}, if we have a set of integrity constraints that include only functional dependencies, the set of all self-inconsistencies equal to the number of maximal independent sets of the conflict graph wherein the tuples of the database are nodes and there are edges between every pair of tuples that violate a functional dependency. So, if you represent the set of integrity constraints that include only functional dependencies as a P4-free graph, then the set is computable in polynomial time.

The measure I_R should also be intractable for functional dependencies. So, this measure should be the size of the minimum vertex cover of the conflict graph. As long as the set of integrity constraints contains one functional dependency per relation, then this too can be computed in polynomial time. Otherwise, the complexity changes.

The Subset Repair System

Since only the rational measures are intractable and almost none of the inconsistency measures end up satisfying all four properties, Livshitz et al. propose a new measure that is both rational and tractable in the case where a constraint is a dependency constraint and operations are tuple deletions. This new measure is described as follows:

Let D be a database, Σ be a finite set of dependency constraints, and let the repair system consist of tuple deletions.

The measure $I_R(\Sigma, D)$ is the result of the Integer Linear Program where each x_i for $i \in ids(D)$, determines whether to delete the i^{th} tuple or not. The solution to the linear relaxation of this Integer Linear Program is a linear program whose last constraint is replaced with "$\forall i \in ids(D) : 0 \leq xi \leq 1$." Denote this as I_R^{lin}.

Comparing this to the previously stated measure I_R, two databases under I_R and I_R^{lin} are consistent with each other if they have sufficient separation under I_R^{lin}. More formally, for two databases D_1, D_2 we have $I_R^{lin}(\Sigma, D_1) >= \mu \cdot I_R(\Sigma, D_2)$, where μ is the integrality gap of the LP relaxation. The maximum number of tuples involved in a violation of a constraint in Σ gives the upper bound on this integrality gap. Finally, this all implies that $I_R^{lin}(\Sigma, D_1) >= I_R^{lin}(\Sigma, D_2)$.

This new theorem is more desirable than I_R for tuple deletions and dependency constraints because it avoids the inherent hardness of I_R while still fulfilling the desired properties.

Experimental Evaluation

To test all the concepts mentioned above, Livshitz et al. used real-world datasets including Stock, Hospital, Food, Airport, Adult, Flight, and voter and the synthetic Tax dataset. They used a dependency constraint mining algorithm to obtain a set of dependency constraints for each dataset. All measurements were done using Python and the Pandas library. All conflicting pairs of tuples were materialized using SQL.

Each experiment was repeated five times and only the average times were reported.

The datasets were all consistent to begin with. To add noise, the authors used two algorithms: Constraint-oriented Noise (CONoise) and Random Noise (RNoise). CONoise introduces random violations of the constraints and RNoise introduces random noise according to a predefined level given by the user.

Results

The behavior of the measures were evaluated on samples of 10,000 tuples from each dataset. First, they ran 200 iterations of the CONoise algorithm to add noise to each dataset and computed the measure values at each iteration. Then, they ran the RNoise algorithm until they modified 1% of the values in the dataset.

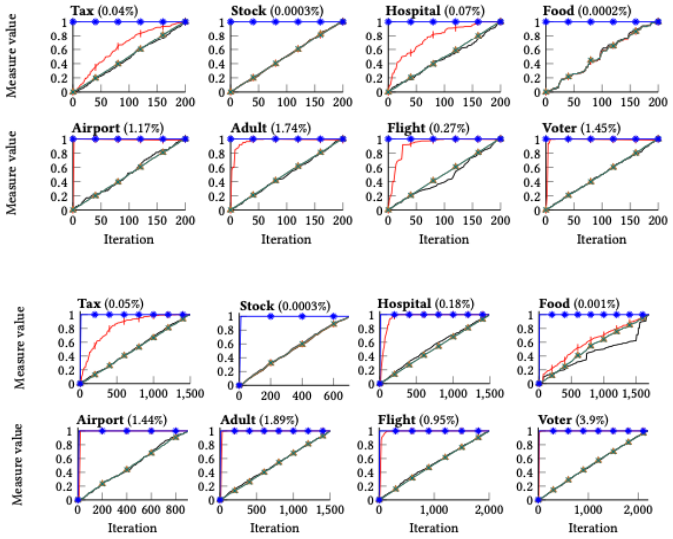
From running variations with the added noise, it was observed that the drastic measure I_d jumps from zero to one when they introduced the first violation. The I_P measure also behaves in a similar way.

The measure I_R is able to recognize that the dataset contains a single erroneous tuple, and reacts to this small change in a proportional way. This is also seen with the I_R, I_R^{lin} , and I_{MI} measures.

The measure I_{MC} incorporates high computational costs so its behavior was tested on a smaller sample. Specifically, the sample was composed of 100 tuples from each dataset. Other than that, the procedure was the same as the previous tests. Results show that this measure is the least stable.

From analyzing the data, the error rate increases with the number of iterations and affects the values of each measure. But it has no effect on actual behavior. Also, the graphs seem similar to each other with varying variables in the noise algorithms. This means that data skew and different distributions of error types do not seem to affect the results. Lastly, to measure the effects of overlap of dependencies, the authors computed the ratio of dependency constraints that overlapped for every dataset and for every dependency constraint. There was no correlation between the behavior of the measures and level of overlap.

The results can be summarized in the following graphs. The first group shows the results with CONoise and the second group shows the results with RNoise:



I_d (—●—), I_{MI} (——), I_P (—+—), I_R (—▲—), and I_R^{lin} (—×—)

Results also show that the measures are mostly stable across several properties of the data and constraints. The error rate does affect the value of the measures but does not affect their behavior. Data skew and different distributions of error types also do not affect the end results as the authors saw that the charts obtained by experiments with different RNoise variables and

typo probabilities were very similar. The graph on the left shows results for CONoise and the graph on the right shows results for RNoise:

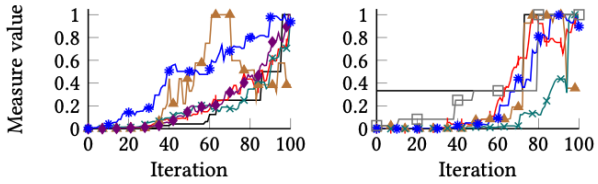
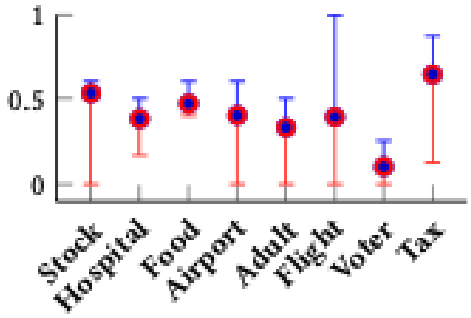


Figure 5: $\mathcal{I}_{\mathrm{MC}}$ (normal.): Stock (——), Hospital (—|—), Food (—□—), Airport (—▲—), Adult (—×—), Flight (—◆—), Voter (—✱—), 100 iterations. Left: CONoise, **Right:** RNoise ($\beta = 0$).

The results for overlap dependencies show that there is no correlation between the measures and the level of overlap between the ratio on denial constraints. The following graphs show the minimum, maximum, and average values for each dataset:



Case Study: The HoloClean Repair System

Finally, to further strengthen the findings stated in the previous section, Livshitz et al. perform similar experiments using the HoloClean system as a comparison. For this experiment, the authors used the Hospital dataset provided by the makers of HoloClean with a set of 15 denial constraints. The performance of HoloClean is already known for this dataset.

To simulate the cleaning pipeline, the authors ran HoloClean on the original dataset with a single denial constraint and then on the resulting dataset after adding one or more denial constraint set. This is repeated multiple times. The same measures as before were measured after every step.

The results from this showed similar results where I_d and I_P fail to indicate progress. The other measures I_R and I_R^{lin} on the other hand, were able to show the reductions in inconsistency levels. The results can be seen in the following plot:

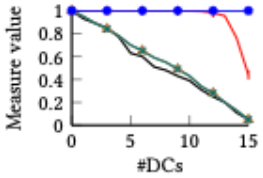


Figure 7: HoloClean case study—normalized measures on Hospital: $\mathcal{I}_d$ (—●—), $\mathcal{I}_{\mathrm{MI}}$ (——), $\mathcal{I}_P$ (—|—), $\mathcal{I}_{\mathcal{R}}$ (—▲—), $\mathcal{I}_{\mathcal{R}}^{\mathrm{lin}}$ (—×—).

In terms of computation time, I_{MC} exceeded the time limit on all datasets. This probably means it is infeasible in this scenario. The times for all other inconsistency measures show stark similarities. This could be because the evaluation of a SQL query takes dominance in large datasets.

For small datasets, the timeout issue was not there. In fact, the execution times were quite fast. It is also revealed that the computations for I_R increase significantly. The results can be summarized in the following tables:

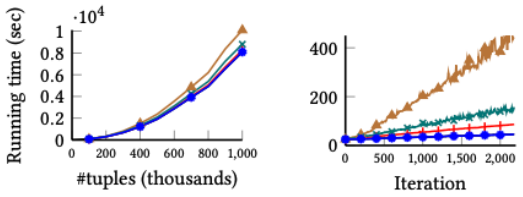


Figure 6: Scalability in $|D|$ on Tax (a) and error rate on Voter (b): $\mathcal{I}_d$ (—●—), $\mathcal{I}_{\mathrm{MI}}$ (——), $\mathcal{I}_P$ (—|—), $\mathcal{I}_{\mathcal{R}}$ (—▲—), and $\mathcal{I}_{\mathcal{R}}^{\mathrm{lin}}$ (—×—).

Conclusion

Livshitz et al. explored the inconsistency measures for databases and their several properties. They make an argument that the right consistency measure depends on the application and dataset. They show that the measures that behave ideally for tuple deletions will also show good empirical behavior, even when models capture attribute updates. They also propose a

new inconsistency measure I_R^{lin} that fulfills all four properties and maintains tractability.