Probabilistic Database Repair Approach based on Answer Set Programming

Anushka Kulkarni[1] **Maurice van Keulen**[2]

[1]Student of EEMCS, University of Twente, Enschede, The Netherlands.
`a.h.kulkarni@student.utwente.nl`

[2]Faculty of EEMCS, University of Twente, Enschede, The Netherlands.
`m.vankeulen@utwente.nl`

Abstract

A database is inconsistent if it violates the integrity constraints defined on it. Many 'non-probabilistic' approaches have been developed to tackle the challenging task of deriving consistent answers from an inconsistent database. While these approaches provide a certain answer, they quite often miss expected results in many cases and have low coverage, recall, and precision. Through this paper, we introduce a new approach called the 'Probabilistic Database Repair Algorithm' based on the 'Theory of Possible Worlds.' In this approach, we consider all possible scenarios for making a database consistent, and we associate each such world with a probability. Therefore, the final result is probabilistic, thus allowing data users to make an informed decision. The approach has a promising coverage of **1** and a recall of **0.7232**. Precision having a value of **0.6066** is comparable to existing methods. However, the probabilistic approach also introduces a noise of **0.3934**, which represents incorrect answer retrieved by the approach. But if systems do tolerate such noises, the probabilistic repair approach proposed is a significant improvement in terms of coverage and recall.

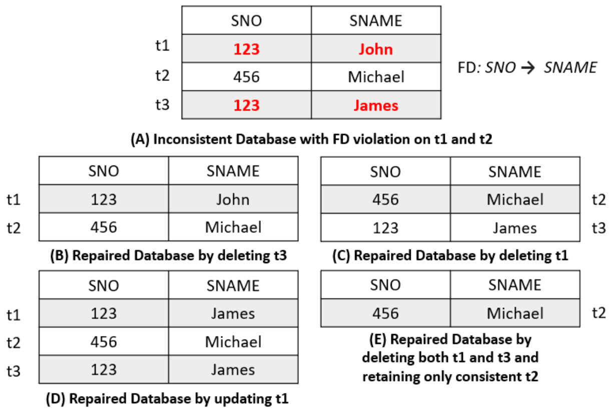
1 Introduction

Data is an essential component of all processes and systems in today's data-driven era; any problems with data quality directly affect the workflows that use it and the outcomes they yield. And we often cannot completely trust the quality of data obtained from a multitude of sources. Such issues could be due to either inconsistencies in the original source data or newly introduced inconsistencies during the process of data integration from multiple sources. Duplication, redundancy, inaccuracies, missing values, and uncertainties in the values are some of the most common yet intricate issues in the source data that must be 'repaired' [1].

For the scope of this paper, we focus only on relational databases, i.e., data in tabular format with a well-defined schema. To identify and repair inconsistencies in such databases, we would need a reference to its consistent state, where we know the data is 'correct.' However, in real-world applications, we often have very little or no information about what the supposedly correct data should be. This knowledge is often limited to having domain experts and human users, which is not always possible and can be expensive. To address this issue, it would be beneficial to establish a 'set of rules' that a relation in a relational database must follow to ensure consistency. In relational databases, such rules are facilitated by Integrity Constraints (ICs) [2]. Constraints are a set of rules defined for a relation to ensure data integrity. Data can be repaired utilizing these constraints such that the repaired database now satisfies all the ICs [3]. In this paper, we focus on one specific type of IC called 'Functional Dependency (FD)' [4; 5; 6]. FDs define the relationship between attributes of a relation. A set of attributes A and B of any relation R are functionally dependent on each other, i.e., $A \rightarrow B$, if values of A can uniquely determine values of B.

Leveraging functional dependencies can fix inconsistent databases by ensuring strict adherence [7]. Several approaches have been proposed in this direction, like the 'update-based repair' approach [10], 'consistent query answering (CQA)' [1; 8; 9], the user input or domain knowledge-based approach [11], argumentation [12], and the answer-set programming approach (ASP) [20; 21], to name a few. Since there are multiple possible ways to make sure that a relation satisfies a given set of constraints, this method leads to multiple possible repairs of an inconsistent database. This is because to ensure that the database satisfies a set of constraints, the underlying data can be modified in different possible ways. So, to maintain consistency, the correct answer to a query posed to a repaired database should be the one that is present in all its repaired 'versions,' i.e., all possible repairs. This means we may not always get a correct or complete answer, or the query result could be an empty set. Further, since we are looking at only one certain answer, we might lose the other possibilities.

Example 1.1. *Consider the database consisting of a 'Student' relation as shown in the Figure 1 (A). The relation has two attributes - (1) SNO ('Student Number' indicating a unique student ID), (2) SNAME ('Student Name'). A functional dependency is defined on the relation: SNO $\rightarrow$ SNAME, which uniquely determines the 'Student Name' of a student, given their 'Student Number'. As indicated, tuples t1 and t3 violate this FD as for a given SNO, SNAME should be unique. This violation makes the database to inconsistent.*


FD: *SNO* → *SNAME*

	SNO	SNAME
t1	123	John
t2	456	Michael
t3	123	James

(A) Inconsistent Database with FD violation on t1 and t2

	SNO	SNAME
t1	123	John
t2	456	Michael

(B) Repaired Database by deleting t3

SNO	SNAME	
456	Michael	t2
123	James	t3

(C) Repaired Database by deleting t1

	SNO	SNAME
t1	123	James
t2	456	Michael
t3	123	James

(D) Repaired Database by updating t1

SNO	SNAME	
456	Michael	t2

(E) Repaired Database by deleting both t1 and t3 and retaining only consistent t2

Figure 1: (A) is an example of FD violation for a relation 'Student' with 'SNO' and 'SNAME' attributes. (B)-(E) are some examples of possible repairs for the 'Student' relation. Here, minimal number of actions to repair the database is 1. So, while (B)-(D) are minimal repairs, (D) is not.

Example 1.1 describes a relation 'Student' and a defined FD on the relation. This relation is inconsistent with respect to the defined FD, as can be seen in Figure 1 (A), and Figures 1 (B)–(E) show possible repairs of the inconsistent relation obtained by modifying the underlying data in a way that satisfies the defined FD. As can be seen from the figure, tuples $t1$ and $t3$ are inconsistent. Let's say we want to retrieve all possible SNAME from the 'Student' relation with SNO = 123. Using the current approaches, we need to determine tuples that are present in the intersection of all the repaired versions. In this case, it's an empty set, which is ideally not desirable.

To overcome these challenges, it is important to consider all possible repairs to avoid data loss and ensure completeness in the answers. In this paper, our goal is to develop an algorithm that repairs an inconsistent database in a way that represents all possible repairs. Additionally, we recognize that not all possible repairs have the same likelihood of being true. Some repairs have a better chance of being true with respect to the given data. Therefore, we propose all possible repairs, each with an associated likelihood, to ensure the retrieval of all complete and consistent answers. This means that all possible answers to a query posed on the database will have an associated probability score, which can be used to make informed decisions. This approach is particularly useful in real-world scenarios where we do not have complete information about the original database; hence, we can only make an estimate of the repaired database. Probability scores in our approach will help choose this best estimate. Our proposed approach is based on modeling the repair as a 'probabilistic problem,' where each possible repair has a linked probability [13; 14; 15; 16; 17; 5; 18; 19].

The paper is structured as follows:

Summary of the main contributions of this paper:

- Developed a probabilistic database repair algorithm to retrieve consistent answers from an inconsistent database. This approach is based on the non-probabilistic ASP algorithm [20; 21] (Section 4)
- Proposed relevant evaluation metrics to assess the performance of the algorithm: expected average precision and recall, precision and recall at a probability P, coverage, and noise (Section 5.2).
- Implemented a variant of the non-probabilistic CQA approach. Result of CQA is considered as baseline in our research [1; 8; 9](Sections 2.1 and 4.2).
- Experimental setup by corrupting a consistent database by introducing inconsistencies that violate the defined FDs. A set of 28 queries is designed to evaluate the results retrieved for both CQA and probabilistic approach. To evaluate, a comparative analysis is performed for the results obtained (Sections 4.1 and 5)

2 Related Work

Repair for an inconsistent database can be seen in two ways: (1) altering the database to make it consistent by actually updating, inserting, or deleting tuples; and (2) querying the original database in a way that retrieves only consistent answers (without actually modifying the original database). In both cases, a consistent answer to a query posed to an inconsistent database is the one that is present in all possible repairs, obtained by making minimal changes to the corrupted database [1; 8; 9]. Here, minimal changes refer to making only sufficient modifications to the underlying data so that the constraints are now satisfied. In Figure 1, repairs (B)–(D) are minimal as they make only one modification, whereas repair (E) is not minimal as it makes two modifications, which is more than necessary. In our work, we consider only such 'minimal' repairs to make the algorithms cost-efficient.

In the literature, we can distinguish two types of repair algorithms: (1) non-probabilistic or (2) probabilistic. Non-probabilistic can either alter the existing database to make it consistent [10; 11; 12] or they query the database in a way that only consistent answers are obtained (without modifying the underlying data) [1; 8; 9]. Current probabilistic algorithms utilize probability distributions to identify the accurate outcome, ensuring consistency in the final response. In this paper, we introduce a second type of probabilistic approach, where the query result obtained is probabilistic, with multiple possibilities each having an associated probability value. Figure 2 illustrates the repair types.

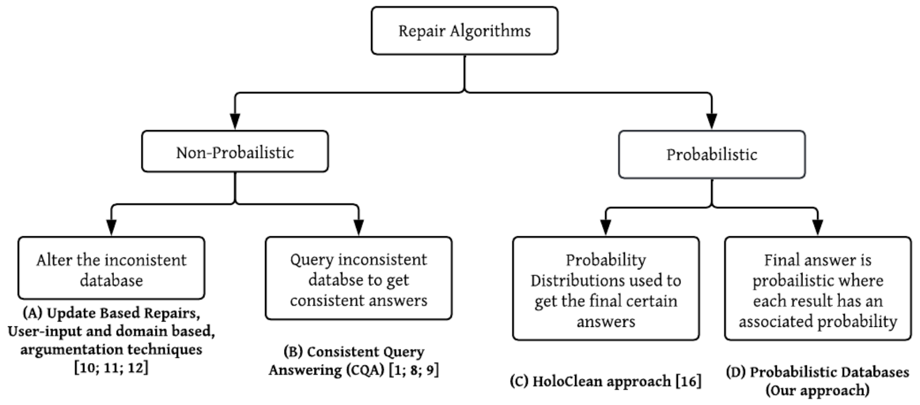

Repair Algorithms
Non-Probailistic
Probabilistic
Alter the inconistent database
(A) Update Based Repairs, User-input and domain based, argumentation techniques [10; 11; 12]
Query inconsistent databse to get consistent answers
(B) Consistent Query Answering (CQA) [1; 8; 9]
Probability Distributions used to get the final certain answers
(C) HoloClean approach [16]
Final answer is probailistic where each result has an associated probability
(D) Probabilistic Databases (Our approach)

Figure 2: Types of repair algorithms.

2.1 Non-Probabilistic Repair Approaches

There are several ways to envision a repair for any inconsistent database D, leading to a large yet finite set of repaired instances. A repaired database D' with respect to D satisfies all the integrity constraints (ICs) defined on D and differs from D in the minimal possible way [10; 22; 9]. This is called the S-repairs approach [9]. A consistent answer, in this case, is the one that appears in all the database repairs D' as depicted by Leopoldo Bertossi [9]. Figure 1 depicts some of the possible corrections. Figure 1 (E) does not depict an S-repair because the changes are not minimal.

On the contrary, an approach called 'Consistent Query Answering (CQA),' [22], argues that rather than actually calculating all the possible repairs to a database and executing the query on the modified database, it is much easier to modify the query itself, which can then be executed on the inconsistent database. This approach is based the concept of Semantic Query Optimization (SQO), which uses the semantics of the database, such as integrity constraints, by appending them to the original query to form an optimized query. Although this approach avoids any potential data loss and corruption, nevertheless, the CQA approach has its limitations in terms of the complexities of calculating the residues, especially for Boolean conjunctive queries (BCQs) and a few functional dependencies (FDs), as discussed in the paper [22].

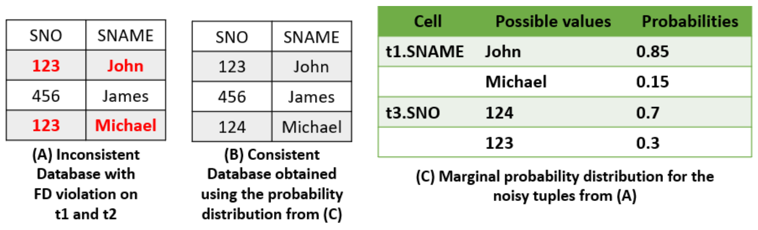
For our proposed algorithm, we take inspiration from one particular type of non-probabilistic repair—the Answer Set Programming (ASP) approach [21; 20; 29; 28]. ASP is used for solving hard computational problems, which often have constraints over binary domains, using a simple and intuitive modeling language. It draws inspiration from knowledge representation, logic programming, and constraint satisfaction. To achieve this, declarative programs are used to define precise steps. These programs are built on atoms and literals (basic facts) and rules connecting them. Results are answers set, which are stable models representing a set of atoms that represent a possible 'world' (solution) satisfying all rules.

All the approaches discussed so far are non-probabilistic in nature. Following section introduces the probabilistic approaches to solve the repair problem.

2.2 Probabilistic repair approach - The HoloClean Framework

One of the most important probabilistic approaches is the 'HoloClean Framework' [16], which is a powerful tool for holistic data repair driven by probabilistic inference. This approach unifies qualitative and quantitative data repair methods by incorporating integrity constraints and the statistical properties of the input data. Moreover, the authors use Bayesian analysis to refine the domain of random variables that correspond to noisy cells in the input dataset. To achieve this, the input dataset is divided into non-overlapping groups of tuples and correlations added by the ICs to tuples in the same group are enumerated. Figure 3 (C) illustrates the calculation of marginal probabilities for each of the noisy tuples. These probabilities determine the correct data values and return them as certain answers. Despite the high average precision and recall reported by HoloClean, the computational complexity of the approach is a significant challenge. Additionally, the final output of HoloClean is a certain update to the database, which might be a wrong update, corrupting the database instead of cleaning it.

Our approach to deriving consistent answers from an inconsistent database is based on the *'Possible Worlds Theory*' of *Probabilistic Data Integration'* [17; 14]. When a repair of any database is performed, we may have little to no knowledge of the real-world data. In all the work done for database repairs so far, there is always a chance that we lose data while performing delete or update actions. In approaches such as Consistent Query Answering (CQA), there is also a risk of yielding no results if all records are included in the potential repairs. We exploit the expressiveness of a probabilistic database to model the uncertainty about which possible repair is the correct one by representing all possible database states with an associated probability. In the following subsection, we formally revisit the theory of possible worlds.



SNO	SNAME
123	John
456	James
123	Michael

(A) Inconsistent Database with FD violation on t1 and t2

SNO	SNAME
123	John
456	James
124	Michael

(B) Consistent Database obtained using the probability distribution from (C)

Cell	Possible values	Probabilities
t1.SNAME	John	0.85
	Michael	0.15
t3.SNO	124	0.7
	123	0.3

(C) Marginal probability distribution for the noisy tuples from (A)

Figure 3: HoloClean repair approach for the inconsistent 'Student' relation from Figure 1 (A)

3 Background and Problem Setup

We introduce a few concepts from our approach in this section, along with the formal notations we will use throughout the paper.

3.1 Existing approach from literature

To evaluate the correctness of our algorithm, we need a baseline to compare it against. For the baseline, we take the existing CQA approach. As discussed in section 2, the CQA approach modifies the queries posed to the inconsistent relation instead of modifying the database itself. The queries are changed in such a way that the modified query ensures the adherence of the defined IC on the relation. The method is depicted in Example 3.1.

Example 3.1. *Consider the inconsistent database from Figure 1 (A).*
Sample SQL query on the relation: ***Select SNO, SNAME from Student****.*
This would return all the tuples t1, t2 and t3, which is not correct as a SNO has to be unique as per the define FD on the relation.

Modified query by appending FD (CQA approach): ***Select SNO, SNAME from Student S1 where NOT EXISTS (Select * from Student S2 where S1.SNAME = S2.SNAME and S1.SNO <> S2.SNO)***
This would return t2 only, as only t2 satisfies the appended condition.

Details of implementation of this approach are in section 5.1.

3.2 Our Approach

We base our approach on two main concepts: 1) Answer Set Programming (ASP) and 2) Possible Worlds Theory. ASP is an existing repair approach from literature that has been discussed briefly in 2.1. We first define the formal notations used in ASP in the following subsection, and then we use the example 3.2 to further explain the algorithm.

3.2.1 Formal Notations: ASP

As defined in [20], let D be a database and let P represent the predicates in the database D. To construct repairs for the database, for each predicate P, a new predicate $P\_$ is introduced. It basically is P augmented with an extra attribute that can take the following values:

- `t`: repair by insertion of the tuple in the new table
- `f`: repair by deletion of the tuple from the new table
- `t*`: tuple was in the original table or inserted in the repair process
- `t**`: tuple belongs to final contents of the new table
- $\mathcal{M}_i$: repair i generated out of all possible repairs

Example 3.2. *Consider the 'Student' relation from Example 1.1 and Figure 1 (A) with FD constraint SNO $\rightarrow$ SNAME. As seen from the figure, the 'Student' table is inconsistent as it violates the defined FD. We now illustrate a repair approach based on ASP algorithm as defined in [20] (Page 189, Section 3.1)*

As per the ASP approach, following repairs would be generated:

Model $\mathcal{M}_1$: *Students_* = {(123, *John*, $\texttt{t}^*$), (456, *Michael*, $\texttt{t}^*$), (123, *James*, `f`), (123, *John*, $\texttt{t}^{**}$), (456, *Michael*, $\texttt{t}^{**}$)}

Model $\mathcal{M}_2$: *Students_* = {(123, *John*, `f`), (456, *Michael*, $\texttt{t}^*$), (123, *James*, $\texttt{t}^*$), (123, *James*, $\texttt{t}^{**}$), (456, *Michael*, $\texttt{t}^{**}$)}

Here, model $\mathcal{M}_1$ represents the repair generated by deleting the tuple (123,James) whereas $\mathcal{M}_2$ represents the repair generated by deleting the tuple (123,John)

3.2.2 Formal Notations: Possible Worlds Theory [14; 17]

The 'Possible Worlds Theory' defines each real-world object using a tuple T (a possibility), and a relation represents a set of real-world objects (set of possibilities). Consequently, the database represents possible worlds. Each tuple t_i is associated with a sentence φ_i, thus defining a tuple entry by (t_i, φ_i).

φ_i is a propositional formula with atoms of the form r = v. For instance, consider the data from Figure 1 (A). The tuple t_1 can be represented as follows:

$$(SNO = 123, SNAME = "John">, X = 1 \wedge Y = 0)$$

.

We define X and Y as random variables, where $X = 1$ and $Y = 0$ denote alternatives for the uncertainties X and Y, respectively. Random variables X and Y can represent any real-world situation. For example, in our relation, X could represent the uncertainty of whether the two tuples $t1$ and $t3$ from Figure 1 (A) are the same, and Y could represent which one of the two alternatives for SNAME is correct. Here, $X = 1$ is one possible assignment, which could mean that the tuples are the same, and $Y = 0$ could be interpreted as the SNAME value of tuple $t1$, which is actually correct. [1] A world w is formed by a random assignment to each of the random variables. A tuple t_i exists in a world if and only if φ_i is satisfied for the random variable assignment.

Possible worlds are essential in probabilistic databases to describe data uncertainties. Each possible world represents a particular data interpretation, encompassing possibilities that might actually exist. This approach helps formulate probabilistic queries for inconsistent or noisy data.

4 Technical Contributions: Our Approach

When querying an inconsistent database, we can realize several possible answers, as the repair is not necessarily unique. All the approaches so far focus on getting only one certain answer, either using a non-probabilistic or a probabilistic approach. Despite its certainty, this perspective has two key drawbacks: (1) the algorithm may return an incorrect possible answer, i.e., it returns one of the possible answers but one that does not match with the real world, and (2) it may lead to data loss. As shown in section 3.1, the CQA approach returns the correct answer to a query as the one that is present in all possible worlds. This means that even if an answer is actually correct in relation to the real world, it can still be discarded if it is not included in the intersection of all possible worlds. On the contrary, our proposed 'Probabilistic Database Repair algorithm' considers all the possible outcomes with a probability, retaining all the possible worlds. The data users can then make an informed decision based on the probability values.

4.1 The Probabilistic Database Repair Algorithm

To explain our repair approach in detail, we first begin by introducing some important concepts and formal notations required for understanding the algorithm. We then present the flow on a high level. Later, we elaborate on individual steps using a running example for better explanation.

4.1.1 Concepts and Formal Notations

D: Relational database which consists of relations that needs to be queried.

R: Inconsistent relation $\in D$

C: Set of columns (attributes) in R

FD_LIST: List of integrity constraints, functional dependencies in this case defined on the relation R. In general, any relation R can have any number and type of ICs defined on it. For the scope of this paper, we only focus on FDs. Each FD in FD_LIST is of the form A $\rightarrow$ B, where A, B is a set of attributes $\in C$

[1]This is just one possible interpretation of the assignment. In actual applications, these definitions need to be specified explicitly to avoid any confusion.

FD_LHS: Left hand side of an FD, i.e, determinant attributes. For example, in A $\rightarrow$ B, A is FD_LHS.

FD_RHS: Right hand side of an FD, i.e, dependent attributes. For example, in A $\rightarrow$ B, B is FD_RHS.

Clusters: Group of tuples (rows) of relation R that have some attributes $\in C$ in common. In our algorithm, we form clusters where FD_LHS, i.e., determinant attributes, are the same. An FD violation essentially means that for the same values of the determinant attribute, the dependent attribute values should also be the same. Hence, forming clusters using FD_LHS helps in identifying and fixing potential violations.

Minimum Actions: To make the algorithm cost-effective and efficient, taking inspiration from S-repairs (section 2), we perform only minimal actions on any inconsistent relation. We limit the scope of actions to updates, deletes, and a combination of updates/deletes only. Here, minimal actions mean performing only the necessary number of updates, deletes, and update/deletes such that the repaired relation now satisfies the defined FD.

Using these concepts and notations, we now present the algorithm in the next section. We will introduce a few other notations in the process.

4.1.2 Flow of the Algorithm

Figure 4 shows the workflow of our probabilistic repair approach on a high level. It takes an inconsistent relation R and a defined list of functional dependencies, FD_LIST, as input and returns a repaired probabilistic database in which each tuple is associated with a probability value. Thereby, instead of one certain answer, any posed query to the repaired database will now give all possible answers, each with an associated probability. These probability values can be used as scores to determine the best possible answers.

In this section, we focus only on Step 2 of Figure 4, as that is the most essential part of the algorithm and has the main logic for performing repair. Other steps are mostly engineering and/or technical steps. The source code for the whole approach and the source code is in Appendix A.

We would like to highlight here that the current algorithm works only for one single FD defined on the relation. Hence, FD_LIST in this case contains only one FD.

Step 2: Clustering & Generating possible repairs per Cluster

- ***Step 2_0: Preprocess data by adding a new column with unique identifier (UID)***
  This is purely a technical step to uniquely identify each row in the relation R. It is needed to make the algorithm space efficient so that we can refer to the stored unique identifier instead of the whole row. This sub-step is especially important because when we generate all possible repairs, we generate all possible combinations, and hence, the rows often repeat. By introducing this preprocessing step, we avoid redundant storage of the whole row by referring to the UID only.

- ***Step 2_1: Identify clusters - group by FD_LHS***
  The idea behind an FD is that all the rows that have the same value for the LHS (determinant) attribute should necessarily have the same values for the RHS (dependent) attribute. This is why in this sub-step we identify rows that have the same value for the FD_LHS attribute, which will belong to the same cluster. Since each cluster now has the same LHS value, we deal with it individually to identify if the RHS value is the same as well. If not, we need to perform a repair.

- ***Step 2_2: Find minimum actions needed within a cluster***
  We now handle one single cluster. Let n be the number of rows in that cluster. We now need to find the minimum number of actions needed to repair that cluster. It is also possible that within a cluster, there are rows that have the same value for the FD_RHS attribute as well. So, we introduce a new variable called k_{max}, which denotes the frequency of the most frequent row in the cluster where values of FD_RHS are the same. Now, if we want to perform minimal actions, logically, these rows with high frequency should be kept constant. This aligns with probability theory, which suggests that if a value appears frequently, it is more likely to be correct. Now, for a minimum number of actions, we can modify all rows except these high frequency rows. Hence,

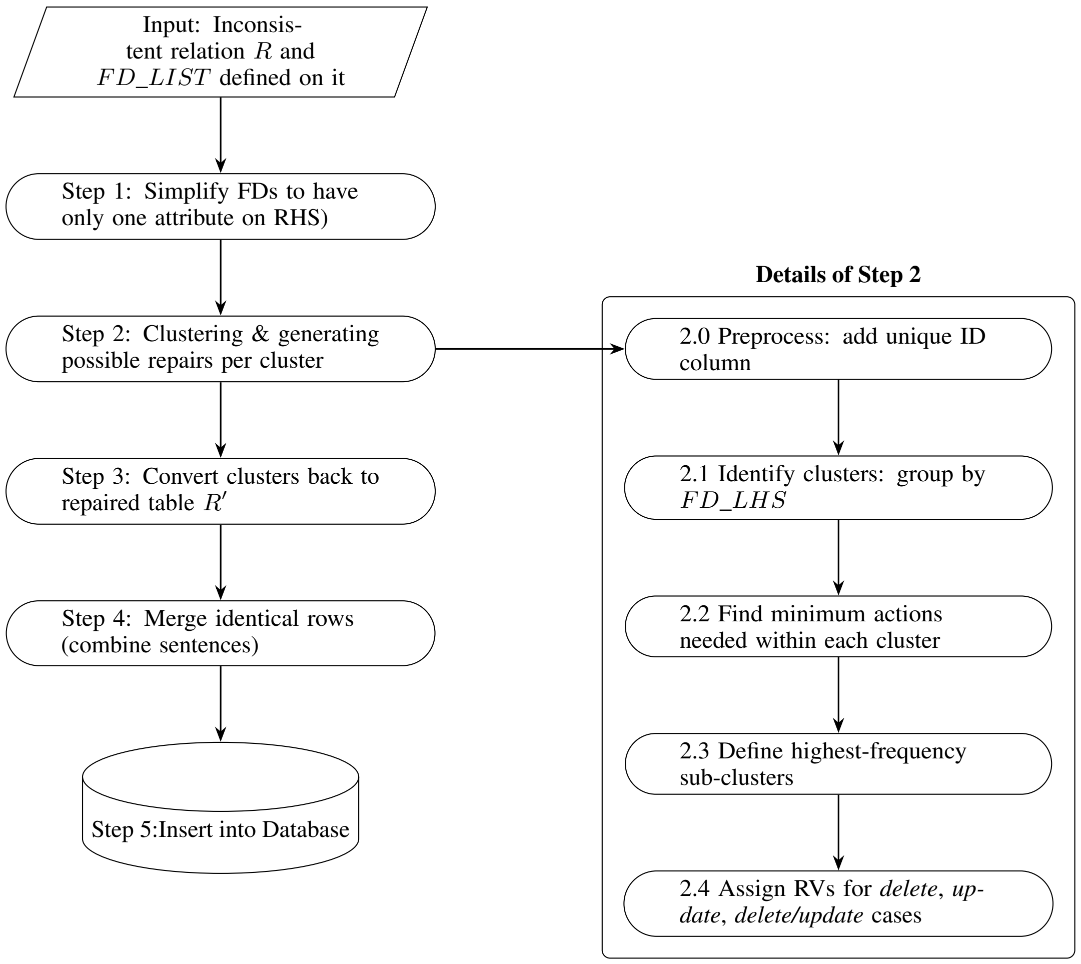

Input: Inconsistent relation R and FD_LIST defined on it
Step 1: Simplify FDs to have only one attribute on RHS)
Step 2: Clustering & generating possible repairs per cluster
Step 3: Convert clusters back to repaired table R'
Step 4: Merge identical rows (combine sentences)
Step 5:Insert into Database
Details of Step 2
2.0 Preprocess: add unique ID column
2.1 Identify clusters: group by FD_LHS
2.2 Find minimum actions needed within each cluster
2.3 Define highest-frequency sub-clusters
2.4 Assign RVs for *delete*, *update*, *delete/update* cases

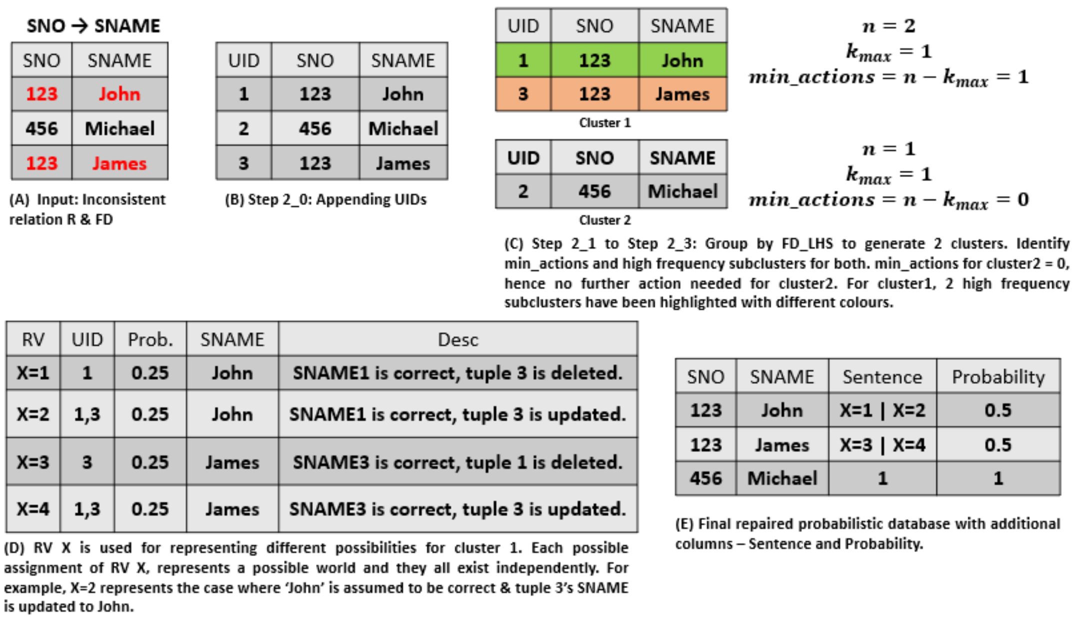
Figure 4: Workflow of the probabilistic repair-generation pipeline

$$min\_actions = n - k_{max} \tag{1}$$

- ***Step 2_3: Define highest frequency sub-clusters***
  Within each cluster, there further can be different sub-clusters each with k_{max} number of rows. These highest frequency sub-clusters are identified for the next step.
- ***Step 2_4: Create and assign RVs for delete, update & delete/update cases***
  For this part of the algorithm, we take inspiration from the 'possible worlds theory' as discussed in section 3.2.2. So, within each cluster, for each possible scenario of update, delete, or update/delete, we assign a value to a random variable (RV). Each of these assignments represent a possible world that exists independently. In this paper, probability values are calculated using a uniform distribution, implying each scenario has an equal probability. This part of the algorithm is explained in Algorithm 4.1.3. We further explain it in detail using a running example in the next section.

4.1.3 Working of the Algorithm: Running Example

To show how our probabilistic database repair approach works, we use the same 'Student' relation from Figure 1 (A) with columns and FD constraint as defined in Example 1.1. The figure reveals that the 'Student' relation is inconsistent due to the presence of tuples $t1$ and $t3$. According to the FD SNO $\rightarrow$ SNAME, for $SNO = 123, SNAME$ should have been the same, which is not the case here.



SNO → SNAME

SNO	SNAME
123	John
456	Michael
123	James

(A) Input: Inconsistent relation R & FD

UID	SNO	SNAME
1	123	John
2	456	Michael
3	123	James

(B) Step 2_0: Appending UIDs

UID	SNO	SNAME
1	123	John
3	123	James

Cluster 1

$n = 2$
$k_{max} = 1$
$min\_actions = n - k_{max} = 1$

UID	SNO	SNAME
2	456	Michael

Cluster 2

$n = 1$
$k_{max} = 1$
$min\_actions = n - k_{max} = 0$

(C) Step 2_1 to Step 2_3: Group by FD_LHS to generate 2 clusters. Identify min_actions and high frequency subclusters for both. min_actions for cluster2 = 0, hence no further action needed for cluster2. For cluster1, 2 high frequency subclusters have been highlighted with different colours.

RV	UID	Prob.	SNAME	Desc
X=1	1	0.25	John	SNAME1 is correct, tuple 3 is deleted.
X=2	1,3	0.25	John	SNAME1 is correct, tuple 3 is updated.
X=3	3	0.25	James	SNAME3 is correct, tuple 1 is deleted.
X=4	1,3	0.25	James	SNAME3 is correct, tuple 3 is updated.

(D) RV X is used for representing different possibilities for cluster 1. Each possible assignment of RV X, represents a possible world and they all exist independently. For example, X=2 represents the case where 'John' is assumed to be correct & tuple 3's SNAME is updated to John.

SNO	SNAME	Sentence	Probability
123	John	X=1 | X=2	0.5
123	James	X=3 | X=4	0.5
456	Michael	1	1

(E) Final repaired probabilistic database with additional columns – Sentence and Probability.

Figure 5: Summary of Probabilistic Database Repair algorithm. Figure (A) shows the input. Here, we just want to focus on Step 2, so we assume that Step 1 has already been performed. Steps (B) through (E) show the sub steps of Step 2 of the main algorithm.

Consider the query: ***Select SNO, SNAME from Student***. For a fair comparison of our algorithm, we compare the result with the traditional non-probabilistic CQA algorithm (section 2). According to Example 3.1, instead of making modifications to the database, CQA appends the constraint to the query to retrieve a consistent answer. Therefore, in our example, CQA retrieves only the tuple t2 for the given query.

We will now follow our algorithm step-by-step using the same query as CQA, applied to the inconsistent database shown in Figure 1.

Input:
$R = [(SNO, SNAME), (123, John), (456, Michael), (123, James)]$
$FD\_LIST = [([SNO], [SName])]$

Here, inconsistent relation R is a list of tuples, where the first tuple contains the attributes of the relation. FD_LIST is a list of tuples, where each tuple consists of two lists. The first list indicates the LHS of the FD, and the second list is the RHS.

Step 1: Simplify FDs to have only one attribute on RHS
No action needed, FDs already have only one attribute as the RHS. This step is a simplification process followed to make the clustering step easier.

Step 2: Clustering & generating possible repairs per cluster

Step 2_0: Preprocess data by adding a new column with unique identifier (UID)
For simplicity in the example, we just add integers 1,2...n as the UID.
$R = [(1, 123, John), (2, 456, Michael), (3, 123, James)]$

Step 2_1: Identify clusters - group by FD_LHS. Group by FD_LHS, i.e., SNO.
$Cluster1 = [(1, 123, John), (3, 123, James)]$
$Cluster2 = [(2, 456, Michael)]$

Step 2_2: Find minimum actions needed within a cluster.
$cluster1$:
$n = 2$
$k_{max} = 1$

$min\_actions = n - k_{max} = 1$

$cluster2$:
$n = 1$
$k_{max} = 1$
$min\_actions = n - k_{max} = 0$

From this step, we focus on $cluster1$ only as no action is needed for $cluster2$.

Step 2_3: Define highest frequency sub-clusters.
For $cluster1$, since $k_{max} = 1$, we have 2 highest frequency sub-clusters,
$subcluster1 = [(1, 123, John)]$
$subcluster2 = [(3, 123, James)]$

Step 2_4: Create and assign RVs for delete, update & delete/update cases
Following Algorithm 4.1.3 line 14-33,

/* UIDs of all rows in cluster */
$all\_set = (1, 3)$

Starting with first highest frequency subcluster, i.e., $subcluster1 = [(1, 123, John)]$
$base\_set = (1)$
$remaining = all\_set - base\_set = (3)$

We now extend the $base\_set$ such that it contains all unique combinations from $remaining$ taking 0 two $min\_actions$ elements from the $remaining$ set at a time. Following combinations are generated:
$(1), (1, 3)$

What this essentially means is the first set represents the case where tuple $t1$ is assumed to be correct, thus deleting tuple $t3$. The second case is where tuple $t1$ is still assumed to be correct, but this time instead of deleting $t3$, we update the value of SNO for $t3$ to make it the same as $t1$. Hence, we use combinations of deletes and updates while maintaining the minimum number of actions. The same process is repeated for the $subcluster2 = [(3, 123, James)]$, generating two more combinations.

Now that we have all possible combinations, we take a random variable, and we assign different values to it. As discussed before, each of these assignments represents a different possible world that exists independently of other worlds. For the scope of this paper, we assume a uniform distribution and hence assign equal probability values to all four cases. The entire process has been summarized in Figure 5. Figures (A) through (C) summarize steps 2_0 to 2_3. Figure (D) shows the RV generation. Here, we choose X as a random variable to define the different possibilities of SNAME. The primary question we aim to address is: which SNAME corresponds to SNO = 123? Hence, X represents these different possibilities.

- ***X=1*** shows the possibility that SNAME = John is correct and that the tuple with UID = 3 should not exist. Essentially, it is the 'delete' cacase,here we remove incorrect tuples.
- ***X=2*** shows the possibility that SNAME = John is still correct, but tuple UID = 3 should also exist. Hence, this shows the 'update' case where the SNAME of tuple 3 is now modified to John as well. While the update doesn't make much sense in this example, as after the update both rows are exactly identical, it is useful in general, as there might be other attributes for the two tuples that still might be different. For example, both UID = 1 and 3 could have an additional attribute called 'Address,' which might be different.

X=3 and X=4 are similar possibilities, but with the assumption that SNAME = James is correct. All these assignments exist independently, and only one of them can be true at a time, hence the probability values for the end users to choose from. Figure 5 (E) shows the final repaired relation. The column 'Sentence' refers to a combination of different RVs. As can be seen from the figure, SNAME = John is true for both X=1 and X=2. Hence, in the final representation, instead of having two separate rows, we merge them by joining the RVs with an 'or.' So, the probability values add up for each 'or,' which then gives the final probability values. For SNO=456, since there are no inconsistencies, the final 'sentence' as well as the probability values are 1, indicating a certain answer. Although this is just a toy example for simplified explanation, it can be understood how the probabilistic database repair approach can be used for repairing inconsistent relations efficiently.

Algorithm 1 Probabilistic Database Repair

```
1: function CREATE_RV_DEFINITIONS(R, FD_RHS, FD_LHS, clusters)
2:     /* Initializing empty dictionaries for rv definitions and rv probabilities */
3:     $rv\_defs = \{\}$
4:     $rv\_probs = \{\}$
5:     /* Iterate over all clusters */
6:     for all cluster $\in$ clusters do
7:         $n = size(cluster)$
8:         find $k_{max}$
9:         $min\_actions = n - k_{max}$
10:        if $min\_actions == 0$ then
11:            /* min_actions = 0 implies no inconsistency in that cluster */
12:            /* Hence no action needed */
13:            continue
14:        else
15:            /* find high_frequency_subclusters */
16:            $all\_set$ = UIDs of all rows in cluster
17:            for all $high\_freq\_subcluster \in high\_frequency\_subclusters$ do
18:                $base\_set$ = UIDs of rows in high frequency subcluster
19:                $remaining = all\_set - base\_set$
20:                for all $i \in range(min\_actions + 1)$ do
21:                    find combinations of $remaining, i$
22:                    for all $combo \in combinations$ do
23:                        $extended\_set = base\_set.union(combo)$
24:                        for all $UID \in extended\_set$ do
25:                            /* Assign RV */
26:                            /* Calculate RV probability */
27:                        end for
28:                    end for
29:                end for
30:            end for
31:        end if
32:    end for
33:    return $rv\_defs, rv\_probs$
34: end function
```

4.2 Probabilistic Database Engineering: DuBio Implementation

The proposed algorithm presents RVs and corresponding probabilities that can be queried using an existing probabilistic database. MYSTIQ [27], Stanford (Trio) [2], and University of Twente's DuBio [3] are a few examples that support probabilistic querying. We use DuBio in our approach, which is a probabilistic variant implemented on top of the Postgres database using Binary Decision Diagrams (BDD) for sentence representation.

[2]http://infolab.stanford.edu/trio

[3]https://github.com/utwente-db/DuBio

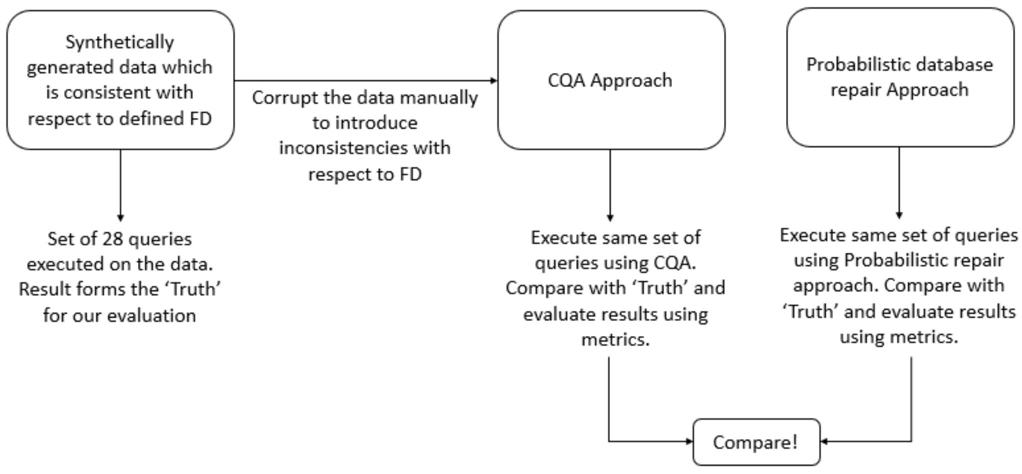

Synthetically generated data which is consistent with respect to defined FD
Corrupt the data manually to introduce inconsistencies with respect to FD
CQA Approach
Probabilistic database repair Approach
Set of 28 queries executed on the data. Result forms the 'Truth' for our evaluation
Execute same set of queries using CQA. Compare with 'Truth' and evaluate results using metrics.
Execute same set of queries using Probabilistic repair approach. Compare with 'Truth' and evaluate results using metrics.
Compare!

Figure 6: Methodology and experimental setup to evaluate the proposed algorithm

5 Methodology

In this section, we present a methodology to evaluate the performance of our algorithm. This process has been summarized in Figure 6.

5.1 Experimental Setup

We first start with a consistent database that satisfies the FD defined there. This data is synthetically generated. Figure 6 shows this step. We use a total of 28 queries, each of which, when executed on the database, gives the 'Truth.' Once 'truth' is obtained, the next step is to introduce inconsistencies in the database so that the algorithms can be evaluated. For this, we randomly manipulate the data so that the defined FD is violated. This forms the inconsistent relation R. We then use the existing CQA approach and our proposed probabilistic database repair algorithm on this inconsistent relation. The answers obtained are then compared with the 'truth' from the first step. The results are evaluated using the evaluation metrics discussed in the next section.

5.2 Evaluation Metrics

After obtaining answers from the 'CQA' and 'Probabilistic database repair' algorithms, we suggest a way to compare them to the 'Truth.' Evaluation metrics are defined in [30]: expected precision, expected recall, and expected noise, and, coverage. For precision and recall, we evaluate 'precision at P' as well as 'recall at P,' which essentially means all answers with a probability $\geq p$ are taken as 1 for the corresponding metric at P. This has been explained in detail in subsequent sections.

Notations and Definitions: [4]

Truth Database	=	TDB	Repaired Database	=	RDB
Expected Answers (EA)	=	$Q(\text{TDB})$	Repaired Answers (RA)	=	$Q(\text{RDB})$
Correct Answers (CA)	=	$\text{EA} \cap \text{RA}$	Wrong Answers (WA)	=	$\text{RA} \setminus \text{EA}$

For the expected values, we take inspiration from the paper [30] and use a similar approach for the calculation of these values.

- ***Expected Recall***: Recall is a measure of how well the algorithm captures all the expected answers for any query executed. It means how many of the expected answers are actually retrieved by the system.

$$\text{Expected Recall} = \frac{\sum_{t \in \text{CA}} P(t)}{|\text{EA}|} \tag{2}$$

[4]Q(D) = Query result obtained by executing query Q on database D

- ***Expected Precision***: Precision is a measure of determining the accuracy of the algorithm. It focuses on how well the system retrieves answers that are actually correct.

$$\text{Expected Precision} = \frac{\sum_{t \in \text{CA}} P(t)}{\sum_{t \in \text{RA}} P(t)} \tag{3}$$

- ***Expected Noise***: Noise is a measure of the inaccuracies predicted by the algorithm. Thus, it determines how wrong the answers predicted by the system are compared to the 'truth.'

$$\text{Expected Noise} = \frac{\sum_{t \in \text{WA}} P(t)}{\sum_{t \in \text{RA}} P(t)} \tag{4}$$

- ***Coverage***: Coverage is very similar to recall, especially for the non-probabilistic CQA approach. This measure still indicates how many of the expected answers the system was able to retrieve, but it does so without considering probability values. Although it is very similar to recall, we still consider this additional measure because we aim to show that even with probabilities, our approach is still much better at retrieving answers compared to the expected answers.

$$\text{Coverage} = \frac{|\text{CA}|}{|\text{EA}|} \tag{5}$$

Here, for the probabilistic database repair approach, each answer is predicted with a probability value. In contrast, for the CQA approach, if an answer is predicted at all, it is always certain. Therefore, for expected precision and recall, summation of probabilities is taken. In the case of CQA, each probability $P(t)$ equals 1 if it is predicted. Hence, for CQA, expected values of precision, recall, and noise are equivalent to calculating it the regular way. However, calculating the expected values this way is necessary for the correct evaluation of the probabilistic approach, as using the expected way as introduced in [30] considers probability values between 0 and 1 instead of assuming it as a certain answer. For recall, the denominator is non-probabilistic, as recall is calculated with respect to the truth database, which is always certain. We next discuss a few additional metrics for the probabilistic approach: precision and recall are measured at a probability P.

- ***Recall at P***: Recall at a probability P essentially calculates recall the regular way as shown in equation 2, the only difference being it considers all the probability values $\geq p$ as 1 and all the probability values p as 0. Therefore, we can claim that assuming most query answers that are wrong have a low probability, at some value of P the recall will still be (almost) 100%.

$$\text{Recall at P} = \frac{\sum_{t \in \text{CA}} P(t)}{|\text{EA}|}, \quad \text{where } P(t) = \begin{cases} 1, & \text{if } P(t) \geq p, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

- ***Precision at p***: Precision at a probability P is defined analogously as recall at p and is given by following equation:

$$\text{Precision at P} = \frac{\sum_{t \in \text{CA}} P(t)}{\sum_{t \in \text{RA}} P(t)}, \quad \text{where } P(t) = \begin{cases} 1, & \text{if } P(t) \geq P, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

- ***F1 at P***: Recall measures how many of the expected answers the system can retrieve, irrespective of other (possibly incorrect) answers retrieved. Precision, on the other hand, measures how accurate the prediction is, i.e., how many answers are correct out of all the retrieved answers. Thus, for a higher precision, it is not just important to get correct answers; it is also important that the percentage of correct answers out of the total answers retrieved is also high. Thus, we need to achieve a balance between precision and recall. The F1 score achieves this balance.

$$\text{F1\_score at } P = \frac{2PR}{P+R}, \quad \text{where } P = \text{Precision at } P \text{ and } R = \text{Recall at } P. \tag{8}$$

Using the evaluation metrics as defined in this section, we compare the results for our proposed approach with CQA using the 'truth' database.

6 Results And Discussions

Using the experimental setup from Section 5.1 and the evaluation metrics defined in Section 5.2, we compute the Expected Recall (Equation 2), Expected Precision (Equation 3), Expected Noise (Equation 4), and, Coverage (Equation 5 for both the CQA and Probabilistic approaches. These results have been formulated in Table 1, followed by a few important observations and their analysis.

Table 1: Comparative Evaluation metrics for expected values

Approach	Recall	Coverage	Precision	Noise
Non-Probabilistic: CQA [20; 8; 9; 22]	0.4214	0.4214	0.6071	0.0
Probabilistic (Our approach)	0.7232	1.0	0.6066	0.3934

- ***Coverage and Recall for CQA are less*** than the probabilistic approach. CQA appends FDs to the original query and executes the modified query on the inconsistent database. So, the modified query will return results only when the appended sub-query is also satisfied. This implies that CQA will give empty results in cases where the FD is violated.
- ***Recall for probabilistic approach is high and Coverage is 1.*** The probabilistic approach is based on the theory of possible worlds, considering every potential scenario for repair. The expected answer is one of all the possible answers, making coverage 100 % in all cases. Also, in cases where CQA does not produce any answers, a probabilistic approach would still provide answers, even though with a possibly low probability. Thus, instead of adding a 0 to the coverage/recall values like CQA for some cases, the probabilistic approach contributes some probability value between 0 and 1, leading to high coverage and recall values.
- ***Recall is same as Coverage for CQA***. CQA returns results only when the appended subquery representing the FD is satisfied. Thus a result would imply that the constraint is satisfied, making the retrieved answer correct. Essentially, coverage and recall are equivalent for CQA because it does not consider probabilistic values; therefore, a correct answer is solely attributed to a value of 1.
- ***Precision is less than coverage pr recall for probabilistic approach***. The probabilistic approach returns all possible answers with an associated probability, and, the probability of a correct answer must be a maximum of 1. Due to the many possibilities, the correct answers only contribute to a fraction of all answers, thus reducing the precision.
- ***Noise is zero for CQA.*** As CQA would either give a correct answer or no answer at all, there is no scenario where it would produce a wrong record in its answer, hence noise is necessarily zero.
- ***Significant noise in probabilistic approach*** Higher noise in probabilistic approach is attributed to the theory of all possible worlds, which considers any combination that might exist. All incorrect answers and their associated probabilities contribute to the noise. Noise is still less because the algorithm would assign lower probability values to possibly incorrect answers.
- ***Almost equal precision of CQA and probabilistic approach*** Precision in CQA is reduced because of empty answers in many of the cases. CQA either provides a correct answer or no answer at all. On the other hand, the precision of the probabilistic approach is reduced because of added 'noise.' As discussed in the previous point, due to the existence of all possible answers with some probability values, these values get distributed amongst precision (correct answers) and noise (incorrect answers). Thus, as far as 'expected precision' is concerned, both the CQA and probabilistic approaches perform equally. Therefore, for a better evaluation of the probabilistic approach, we also calculate precision and recall values at a probability of P.

From the observations so far, we can see that while CQA has zero noise, it also has significantly lesser recall, coverage, and precision. On the other hand, despite some noise, the probabilistic repair method offers significantly higher recall and coverage. In systems that require 100% accuracy but can tolerate missing data, CQA querying would be more effective. However, the probabilistic repair approach would be an ideal choice where a slight noise is tolerable while expecting much greater coverage and recall. Furthermore, since a probabilistic approach provides all possibilities with associated probabilities, it also provides the data user a chance to make an informed decision while choosing the result.

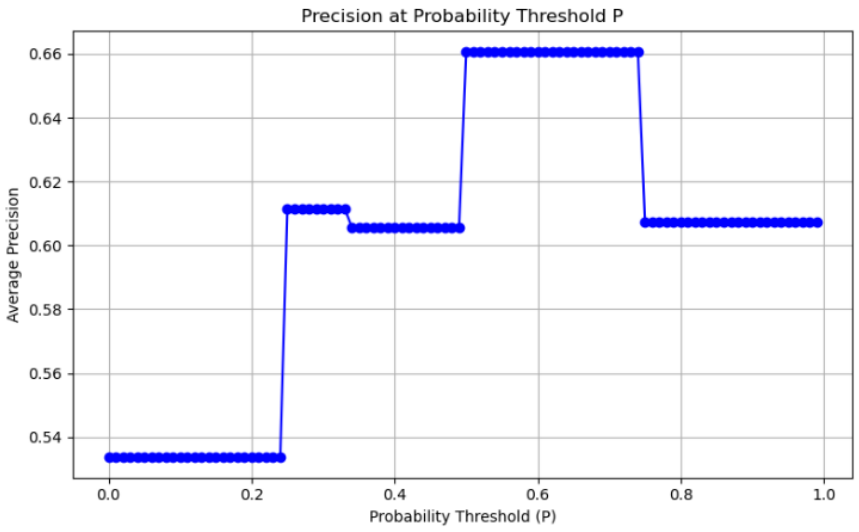


Figure 7: Precision values at a probability of P. X axis here shows the probability thresholds and Y axis shows the precision value. Thus the graph shows the trend of precision for probabilistic approach with increasing probability values.

For the probabilistic repair approach, we also evaluate the precision and recall values at a probability threshold of P. The result has been plotted in Figures 7 and 8. Consider Figure 7; as expected, with higher probability values, precision increases. This is because it is assumed that the approach would assign higher probability values to possibly correct answers. But, we also see an unexpected dip in the precision value with very high probability. This fluctuation can be attributed to the fact that for the scope of this paper, we assume a uniform distribution of probability. This assumption might not be necessarily correct in real-world applications. For instance, in our example from Figure 5, both tuples t_1 and t_3 are assigned equal probabilities of 0.5. But, in reality, it is possible that SNAME 'John' of tuple t_1 is slightly more probable than that of tuple t_3. So far, we do not consider this possible scenario. This is also one of the reasons for low expected precision and slight noise. Referring to Figure 8, recall is higher for lower probability values because all correct answers meet the minimum threshold probability. But as the probability value increases, fewer and fewer correct answers are able to cross the threshold. This can be attributed to a similar argument of lower precision value discussed before. As can be seen, while precision has a maximum value for the probability threshold between 0.5 and 0.75, recall has a maximum value for quite low probability. Thus, to achieve a trade-off, we plot the F1 values as seen in Figure 9. From the figure, our probability threshold is quite low, which is currently a limitation of the algorithm, which is further discussed in detail in the next section.

7 Limitations and Future Research

With the trade-offs seen in Section 6, the probabilistic database approach is quite promising and can be further developed. We discover a few interesting areas of open problems along these lines. Firstly, the current probabilistic approach focuses on only one type of integrity constraint: functional dependency, and that too with only one constraint at a time. This can further be developed to include other constraints like domain, entity, referential, and key to create a more generic model. Also, probability values are currently assumed to be uniform and are derived statistically considering a small sample dataset. With a much larger database and some other computations, probability values can be improved for much better results. Further, we consider a semi-open world assumption, that is, tuples are formed using the values within the database by performing updates and/or deletions. With a fully open world assumption, where one can expect values from outside the dataset as well, edge cases can be better handled. Coming to the F1 score at a probability threshold P, as F1 balances the trade-offs between precision and recall, the probability threshold is quite low because of the very low probability threshold for recall. This can be improved by better probability estimations. Finally, currently, queries like 'Group By' are difficult to model in a probabilistic database due to the

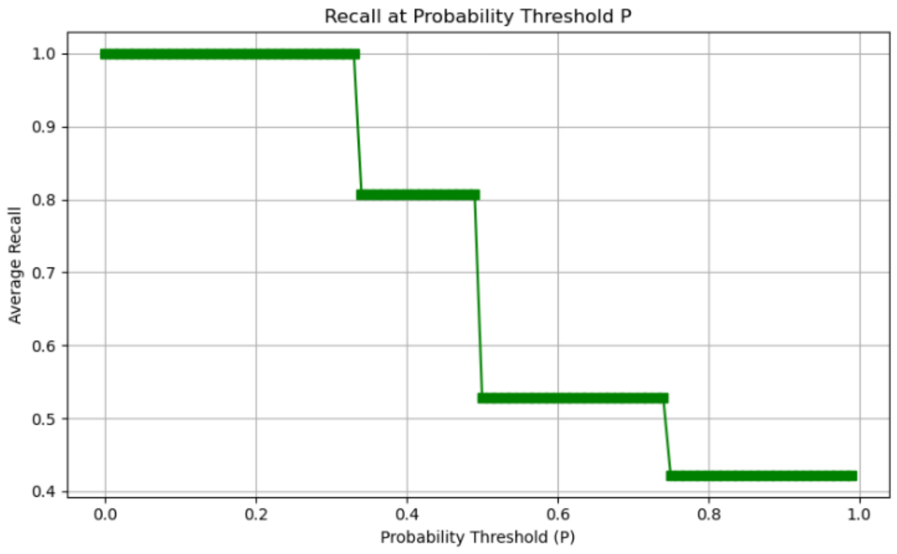


Figure 8: Recall values at a probability of P. X axis here shows the probability thresholds and Y axis shows the recall value. Thus the graph shows the trend of recall for probabilistic approach with increasing probability values.

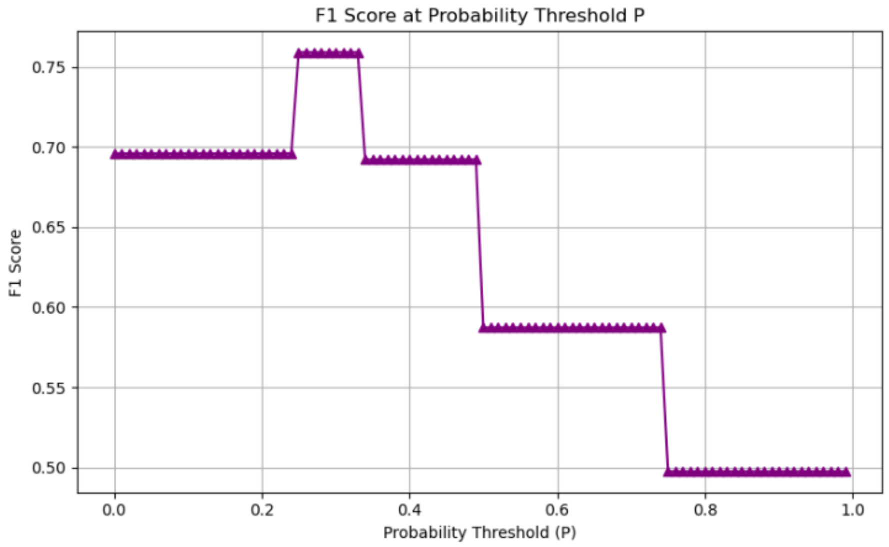


Figure 9: F1 values at a probability of P. X axis here shows the probability thresholds and Y axis shows the F1 value. Thus the graph shows the trend of F1 for probabilistic approach with increasing probability values.

many possible worlds consideration. Finding a way to query such scenarios would be a fine area of research.

8 Conclusions

Through the research presented in this paper, we introduced a 'Probabilistic Database Repair' algorithm to derive consistent answers from an inconsistent database. This approach is based on the Answer Set Programming (ASP) approach that simplifies a problem using small declarative programs. The probabilistic approach, based on the theory of possible worlds and integrity constraints, considers each possible result for a given query and associates all such answers with their corresponding probabilities. It is observed that the coverage and recall for the probabilistic approach are quite high, implying all possible correct answers are retrieved using a probabilistic approach. Precision, on the other hand, with a value of **0.6066**, although less than the coverage, is still decent and comparable to the traditional non-probabilistic CQA approach. Furthermore, It is noteworthy that while recall and coverage are significantly higher in a probabilistic approach, it also introduces noise into the system, having a value of **0.3934**, representing the unnecessary possible worlds retrieved by the algorithm. We further covered some open issues and research opportunities in our approach. While promising, it can be improved by adding integrity constraints, enhancing probability computations with larger datasets, and handling complicated queries like 'Group By.'

References

[1] Bertossi, L. (2011). Database Repairing and Consistent Query Answering. In Synthesis lectures on data management. `https://doi.org/10.1007/978-3-031-01883-1`

[2] Fan, W., Geerts, F., & Jia, X. (2008). A revival of integrity constraints for data cleaning. Proceedings of the VLDB Endowment, 1(2), 1522–1523. `https://doi.org/10.14778/1454159.1454220`

[3] Weber, D. (2017, January 1). A Constraint-Based Approach to Data Quality in Information Systems. `https://doi.org/10.3929/ethz-b-000198644`

[4] Guo, Z., & Rekatsinas, T. (2019). Learning Functional Dependencies with Sparse Regression. arXiv (Cornell University). `https://doi.org/10.48550/arxiv.1905.01425`

[5] De, S., & Kambhampati, S. (2010b). Defining and Mining Functional Dependencies in Probabilistic Databases. arXiv (Cornell University). `https://doi.org/10.48550/arxiv.1005.4714`

[6] Using functional dependency thresholding to discover functional dependencies for data cleaning - University of Twente Student Theses. (n.d.). `https://purl.utwente.nl/essays/85699`

[7] Nofar Carmeli, Martin Grohe, Benny Kimelfeld, Ester Livshits, and Muhammad Tibi. Database Repairing with Soft Functional Dependencies. In 24th International Conference on Database Theory (ICDT 2021). Leibniz International Proceedings in Informatics (LIPIcs), Volume 186, pp. 16:1-16:17, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2021) `https://doi.org/10.4230/LIPIcs.ICDT.2021.16`

[8] Leopoldo Bertossi. 2006. Consistent query answering in databases. SIGMOD Rec. 35, 2 (June 2006), 68–76. `https://doi.org/10.1145/1147376.1147391`

[9] Leopoldo Bertossi. 2019. Database Repairs and Consistent Query Answering: Origins and Further Developments. In Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '19). Association for Computing Machinery, New York, NY, USA, 48–58. `https://doi.org/10.1145/3294052.3322190`

[10] Jef Wijsen. 2005. Database repairing using updates. ACM Trans. Database Syst. 30, 3 (September 2005), 722–768. `https://doi.org/10.1145/1093382.1093385`

[11] Joeri Rammelaere and Floris Geerts. 2018. Explaining repaired data with CFDs. Proc. VLDB Endow. 11, 11 (July 2018), 1387–1399. `https://doi.org/10.14778/3236187.3236193`

[12] Santos, E., Martins, J. P., & Galhardas, H. (2010). An Argumentation-based Approach to Database Repair. European Conference on Artificial Intelligence, 125–130. https://doi.org/10.3233/978-1-60750-606-5-125

[13] Nilesh Dalvi, Christopher Ré, and Dan Suciu. 2009. Probabilistic databases: diamonds in the dirt. Commun. ACM 52, 7 (July 2009), 86–94. https://doi.org/10.1145/1538788.1538810

[14] Wanders, B., & Van Keulen, M. (2015b). Revisiting the formal foundation of Probabilistic Databases. Advances in Intelligent Systems Research/Advances in Intelligent Systems Research. https://doi.org/10.2991/ifsa-eusflat-15.2015.43

[15] Fabian Panse, Maurice van Keulen, and Norbert Ritter. 2013. Indeterministic Handling of Uncertain Decisions in Deduplication. J. Data and Information Quality 4, 2, Article 9 (March 2013), 25 pages. https://doi.org/10.1145/2435221.2435225

[16] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: holistic data repairs with probabilistic inference. Proc. VLDB Endow. 10, 11 (August 2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[17] Van Keulen, M. (2018). Probabilistic Data Integration. In Springer eBooks (pp. 1–9). https://doi.org/10.1007/978-3-319-63962-8_18-1

[18] Matteo Magnani and Danilo Montesi. 2010. A Survey on Uncertainty Management in Data Integration. J. Data and Information Quality 2, 1, Article 5 (July 2010), 33 pages. https://doi.org/10.1145/1805286.1805291

[19] P. Andritsos, A. Fuxman and R. J. Miller, "Clean Answers over Dirty Databases: A Probabilistic Approach," 22nd International Conference on Data Engineering (ICDE'06), Atlanta, GA, USA, 2006, pp. 30-30. https://doi.org/10.1109/ICDE.2006.35

[20] Handbook of Data Quality. (2013). In Springer eBooks. https://doi.org/10.1007/978-3-642-36257-6

[21] Arenas, M., Bertossi, L., & Chomicki, J. (2003). Answer sets for consistent query answering in inconsistent databases. In TLP (Vol. 3, pp. 393–424) [Journal-article]. https://doi.org/10.1017/S1471068403001832

[22] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '99). Association for Computing Machinery, New York, NY, USA, 68–79. https://doi.org/10.1145/303976.303983

[23] Lopatenko, A., & Bertossi, L. E. (2016). Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics (extended version). arXiv (Cornell University). https://doi.org/10.48550/arxiv.1605.07159

[24] Paolo Guagliardo and Leonid Libkin. 2017. Correctness of SQL Queries on Databases with Nulls. SIGMOD Rec. 46, 3 (September 2017), 5–16. https://doi.org/10.1145/3156655.3156657

[25] Bertossi, L. (2017). Specifying and Computing Causes for Query Answers in Databases via Database Repairs and Repair Programs. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1712.01001

[26] Martin Grohe and Peter Lindner. 2019. Probabilistic Databases with an Infinite Open-World Assumption. In Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '19). Association for Computing Machinery, New York, NY, USA, 17–31. https://doi.org/10.1145/3294052.3319681

[27] Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Shobhit Mathur, Chris Re, and Dan Suciu. 2005. MYSTIQ: a system for finding more answers by using probabilities. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data (SIGMOD '05). Association for Computing Machinery, New York, NY, USA, 891–893. https://doi.org/10.1145/1066157.1066277

[28] Brewka, G., Eiter, T., & Truszczyński, M. (2011). Answer set programming at a glance. Communications of the ACM, 54(12), 92–103. https://doi.org/10.1145/2043174.2043195

[29] Barceló, P., Bertossi, L., & Bravo, L. (2003). Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In Lecture notes in computer science (pp. 7–33). https://doi.org/10.1007/3-540-36596-6_2

[30] Van Keulen, Maurice & de Keijzer, Ander. (2009). Qualitative effects of knowledge rules and user feedback in probabilistic data integration. VLDB J.. 18. 1191-1217. https://doi.org/10.1007/s00778-009-0156-z

A Source Code

The repository is available at: https://gitlab.utwente.nl/s3291030/repair-approach-prototype