Consistent Answers of Aggregation Queries via SAT

Akhil A. Dixit
University of California Santa Cruz
akadixit@ucsc.edu

Phokion G. Kolaitis
University of California Santa Cruz and IBM Research
kolaitis@ucsc.edu

***Abstract*—The framework of database repairs and consistent answers to queries is a principled approach to managing inconsistent databases. We describe the first system able to compute the consistent answers of general aggregation queries with the `COUNT(`A`)`, `COUNT(*)`, and `SUM` operators, and with or without grouping constructs. Our system uses reductions to optimization versions of Boolean satisfiability (SAT) and then leverages powerful SAT solvers. We carry out an extensive set of experiments on both synthetic and real-world data that demonstrate the usefulness and scalability of this approach.**

I. Introduction

The framework of database repairs and consistent query answering, introduced by Arenas, Bertossi, and Chomicki [1], is a principled approach to managing inconsistent databases, i.e., databases that violate one or more integrity constraints on their schema. In this framework, inconsistencies are handled at query time by considering all possible *repairs* of the inconsistent database, where a repair of an inconsistent database $\mathcal{I}$ is a consistent database $\mathcal{J}$ that differs from $\mathcal{I}$ in a "minimal" way. The *consistent answers* to a query q on a given database $\mathcal{I}$ is the intersection of the results of q applied on each repair of $\mathcal{I}$. Thus, a consistent answer provides the guarantee that it will be found no matter on what repair the query has been evaluated. Computing the consistent answers can be an intractable problem, because an inconsistent database may have exponentially many repairs. In particular, computing the consistent answers of a fixed Select-Project-Join (SPJ) query can be a coNP-complete problem. By now, there is an extensive body of work on the complexity of consistent answers for SPJ queries (see Section II).

Range Semantics: Concept and Motivation. Aggregation queries, the most frequently asked queries, are of the form

$Q :=$ `SELECT` $Z, f(A)$ `FROM` $T(U, Z, A)$ `GROUP BY` Z,

where $f(A)$ is one the standard aggregation operators `COUNT(`A`)`, `COUNT(*)`, `SUM(`A`)`, `AVG(`A`)`. `MIN(`A`)`, `MAX(`A`)`, and $T(U, Z, A)$ is the relation returned by a SPJ query q expressed in SQL. A *scalar aggregation* query is an aggregation query without a `GROUP BY` clause.

What is the semantics of an aggregation query over an inconsistent database? Since an aggregation query may return different answers on different repairs of an inconsistent database, there is typically *no* consistent answer as per the earlier definition of consistent answers. To obtain meaningful semantics to aggregation queries, Arenas et al. [2] introduced the *range consistent answers*.

Let Q be a scalar aggregation query and Σ be a set of integrity constraints. The set of *possible answers* to Q on an inconsistent instance $\mathcal{I}$ w.r.t. Σ is the set of the answers to Q over all repairs of $\mathcal{I}$ w.r.t. Σ, i.e., $\text{Poss}(Q, \Sigma) = \{Q(\mathcal{J}) \mid \mathcal{J} \text{ is a repair of } \mathcal{I} \text{ w.r.t. } \Sigma\}$. The *range consistent answers* to Q on $\mathcal{I}$ is the interval $[glb(Q, \mathcal{I}), lub(Q, \mathcal{I})]$, where the endpoints of this interval are, respectively, the greatest lower bound (glb) and the least upper bound (lub) of the set $\text{Poss}(Q, \Sigma)$ of possible answers to Q on $\mathcal{I}$. For example, the range consistent answers to the query

```
SELECT SUM(ACCOUNTS.BAL) FROM ACCOUNTS,
CUSTACC WHERE ACCOUNTS.ACCID = CUSTACC.ACCID
AND CUSTACC.CID = 'C2'
```

on the instance in Table I is the interval $[900, 2200]$. The meaning is that no matter how the database $\mathcal{I}$ is repaired, the answer to the query is guaranteed to be in the range between 900 and 2200. Arenas et al. [3] focused on scalar aggregation queries only. Fuxman, Fazli, and Miller [4] extended the notion of range consistent answers to aggregation queries with grouping (see Section III).

Range semantics have become the standard semantics of aggregation queries in the framework of database repairs (see [5, Section 5.6]). Furthermore, range semantics have been adapted to give semantics to aggregation queries in several other contexts, including data exchange [6] and ontologies [7]. Finally, range semantics have been suggested as an alternative way to overcome some of the issues arising from SQL's handling of null values [8].

Earlier Systems for Consistent Query Answering. Several academic prototype systems for consistent query answering have been developed [3], [9], [10], [4], [11], [12], [13], [14], [15], [16]. These systems use different approaches, including logic programming [9], [12], compact representations of repairs [17], or reductions to solvers [14], [13], [16]. In particular, in [16], we reported on CAvSAT, a system that at that time was able to compute the consistent answers of unions of SPJ queries w.r.t. denial constraints (which include functional dependencies as a special case) via reductions to SAT solvers. Among all these systems, however, only the ConQuer system by Fuxman et al. [4], [11] is capable of handling aggregation queries. Actually, ConQuer can only handle a restricted class of aggregation query, namely, those aggregation queries w.r.t. key constraints for which the underlying SPJ query belongs to the class called C_{forest}. For such a query Q, the range consistent answers of Q are SQL-rewritable, which means that there is a

2022 IEEE 38th International Conference on Data Engineering (ICDE) | 978-1-6654-0883-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDE53745.2022.00074

2375-026X/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDE53745.2022.00074

Table I: Running example – an inconsistent database instance of bank account records

Customer

	CID	NAME	CITY
f_1	C1	John	LA
f_2	C2	Mary	LA
f_3	C2	Mary	SF
f_4	C3	Don	SF
f_5	C4	Jen	LA

Accounts

	ACCID	TYPE	CITY	BAL
f_6	A1	Check.	LA	900
f_7	A2	Check.	LA	1000
f_8	A3	Saving	SJ	1200
f_9	A3	Saving	SF	-100
f_{10}	A4	Saving	SJ	300

CustAcc

	CID	ACCID
f_{11}	C1	A1
f_{12}	C2	A2
f_{13}	C2	A3
f_{14}	C3	A4

SQL query Q' such that the range consistent answers of Q on an instance $\mathcal{I}$ can be obtained by directly evaluating Q' on $\mathcal{I}$. This leaves out, however, many aggregation queries, including all aggregation queries whose range consistent answers are not SQL-rewritable or are NP-hard to compute. Up to now, no system supports such queries.

Summary of Contributions. In this paper, we report on and evaluate the performance of AggCAvSAT (Aggregate Consistent Answers via Satisfiability Testing), which is an enhanced version of CAvSAT and is also the first system that is capable to compute the range consistent answers to all aggregation queries involving the operators `SUM(`A`)`, `COUNT(`A`)`, or `COUNT(*)` with or without grouping.

We first corroborate the need for a system that goes well beyond ConQuer by showing that there is an aggregation query Q involving `SUM(`A`)` such that the consistent answers of the underlying SPJ query q w.r.t. key constraints are SQL-rewritable, but the range consistent answers of Q are NP-hard (Theorem III.1 in Section III).

The distinctive feature of AggCAvSAT is that it uses polynomial-time reductions to reduce the range consistent answers of aggregation queries to optimization variants of Boolean Satisfiability (SAT), such as Partial MaxSAT and Weighted Partial MaxSAT. These reductions, described in Sections IV and V, are natural but are much more sophisticated than the reductions used in [16] to reduce the consistent answers of SPJ queries to SAT. After the reductions have been carried out, AggCAvSAT deploys powerful SAT solvers, such as the MaxHS solver [18], to compute the range consistent answers of aggregation queries. Furthermore, AggCAvSAT can handle databases that are inconsistent not only w.r.t. key constraints, but also w.r.t. arbitrary *denial* constraints, a much broader class of constraints.

An extensive experimental evaluation of AggCAvSAT is reported in Section VI. We carried out a suite of experiments on both synthetic and real-word databases, and for a variety of aggregation queries with and without grouping. The synthetic databases were generated using two different methods: (a) the `DBGen` tool of TPC-H was used to generate consistent data and then inconsistencies were injected artificially; (b) the PDBench inconsistent database generator from the probabilistic database management system MayBMS [19] was used. The experiments demonstrated the scalability of AggCAvSAT along both the size of the data and the degree of inconsistency in the data. Note that AggCAvSAT was also competitive in comparison to ConQuer (especially when the degree of inconsistency was not excessive), even though the latter is tailored to only handle a restricted class of aggregation queries whose range consistent answers are SQL-rewritable.

An extended version of this paper is posted on arXiv [20].

Consistent Answers vs. Data Cleaning. There is a large body of work on managing inconsistent databases via data cleaning. There are fundamental differences between the framework of the consistent answers and the framework of data cleaning (see [5, Section 6]). In particular, the consistent answers provide the guarantee that each such answer will be found no matter on which repair the query at hand is evaluated, while data cleaning provides no similar guarantee. Data cleaning has the attraction that it produces a single consistent instance but the process need not be deterministic and the instance produced need not even be a repair (i.e., it need not be a maximally consistent instance). Recent data cleaning systems, such as HoloClean [21] and Daisy [22], [23], produce a probabilistic database instance as the output (that need not be a repair). At the performance level, the data cleaning approaches remove inconsistencies in the data offline, hence the time-consuming tasks are done prior to answering the queries; in contrast, systems for consistent query answering work online.

It is an interesting project, left for future research, to develop a methodology and carry out a fair comparison on a level playing field between systems for data cleaning and systems for consistent query answering.

II. Preliminaries

Integrity Constraints and Database Queries: A relational database schema $\mathcal{R}$ is a finite collection of relation symbols, each with a fixed positive integer as its arity. The attributes of a relation symbol are names for its columns; they can be identified with their positions, thus $Attr(R) = \{1, ..., n\}$ denotes the set of attributes of R. An $\mathcal{R}$-*instance* is a collection $\mathcal{I}$ of finite relations $R^{\mathcal{I}}$, one for each relation symbol R in $\mathcal{R}$. An expression of the form $R^{\mathcal{I}}(a_1, ..., a_n)$ is a *fact* of the instance $\mathcal{I}$ if $(a_1, ..., a_n) \in R^{\mathcal{I}}$. A *key* is a minimal subset X of $Attr(R)$ such that the functional dependency $X \rightarrow Attr(R)$ holds. The attributes in X are called the *key attributes* of R and they are denoted by underlining their corresponding positions; thus, $R(\underline{A, B}, C)$ denotes that the attributes A and B form a key of R.

First-order logic has been successfully used as a database query language [24]; in fact, it forms the core of SQL. A

conjunctive query is expressible by a first-order formula of the form $q(\mathbf{z}) := \exists \mathbf{w}\ (R_1(\mathbf{x}_1) \wedge ... \wedge R_m(\mathbf{x}_m))$, where each $\mathbf{x}_i$ is a tuple of variables and constants, $\mathbf{z}$ and $\mathbf{w}$ are tuples of variables with no variable in common, and the variables in $\mathbf{x}_1, ..., \mathbf{x}_m$ appear in exactly one of the tuples $\mathbf{z}$ and $\mathbf{w}$. A conjunctive query with no free variables (i.e., all variables are existentially quantified) is a boolean query, while a conjunctive query with k free variables in $\mathbf{z}$ is a k-ary query. Conjunctive queries are also known as *select-project-join* (SPJ) queries with equi-joins, and are among the most frequently asked queries. For example, on the instance $\mathcal{I}$ from Table I, the binary conjunctive query $q(z, x) := \exists w\ (\texttt{CUST}(w, x, y) \wedge \texttt{CUSTACC}(w, z))$ returns the set of all pairs (z, x) such that z is an account ID of an account owned by customer named x.

Database Repairs and Consistent Answers: Let Σ be a set of integrity constraints on a database schema $\mathcal{R}$. An $\mathcal{R}$-instance $\mathcal{I}$ is *consistent* if $\mathcal{I} \models \Sigma$, i.e., $\mathcal{I}$ satisfies every constraint in Σ; otherwise, $\mathcal{I}$ is *inconsistent*. For example, let $\mathcal{I}$ be the instance depicted in Table I. There are two key constraints, namely, CUST(CID) and ACC(ACCID). Clearly, $\mathcal{I}$ is inconsistent since the facts f_2, f_3 of CUST and facts f_8, f_9 of ACC violate these key constraints.

A *repair* of an inconsistent instance $\mathcal{I}$ w.r.t. Σ is a consistent instance $\mathcal{J}$ that differs from $\mathcal{I}$ in a "minimal" way. Different notions of minimality give rise to different types of repairs (see [5] for a survey). Here, we focus on *subset repairs*, the most extensively studied type of repairs. An instance $\mathcal{J}$ is a *subset repair* of an instance $\mathcal{I}$ if $\mathcal{J}$ is a maximal consistent subinstance of $\mathcal{I}$, that is, $\mathcal{J} \subseteq \mathcal{I}$ (where $\mathcal{I}$ and $\mathcal{J}$ are viewed as sets of facts), $\mathcal{J} \models \Sigma$, and there is no instance $\mathcal{J}'$ such that $\mathcal{J}' \models \Sigma$ and $\mathcal{J} \subset \mathcal{J}' \subset \mathcal{I}$. Arenas et al. [1] used repairs to give rigorous semantics to query answering on inconsistent databases. Specifically, assume that q is a query, $\mathcal{I}$ is an $\mathcal{R}$-instance, and $\mathbf{t}$ is a tuple of values. We say that $\mathbf{t}$ is a *consistent answer* to q on $\mathcal{I}$ w.r.t. Σ if $\mathbf{t} \in q(\mathcal{J})$, for every repair $\mathcal{J}$ of $\mathcal{I}$. We write $\textsc{Cons}(q, \mathcal{I}, \Sigma)$ to denote the set of all *consistent answers* to q on $\mathcal{I}$ w.r.t. Σ, i.e.,

$\textsc{Cons}(q, \mathcal{I}, \Sigma) = \bigcap \{q(\mathcal{J}) : \mathcal{J}$ is a repair of $\mathcal{I}$ w.r.t. $\Sigma\}$.

If Σ is a fixed set of integrity constraints and q is a fixed query, then the main computational problem associated with the consistent answers is: given an instance $\mathcal{I}$, compute $\textsc{Cons}(q, \mathcal{I}, \Sigma)$; we write $\textsc{Cons}(q, \Sigma)$ to denote this problem. If q is a boolean query, then computing the consistent answers becomes the decision problem $\textsc{Certainty}(q, \Sigma)$: given an instance $\mathcal{I}$, is q true on every repair $\mathcal{J}$ of $\mathcal{I}$ w.r.t. Σ? When the constraints in Σ are understood from the context, we will write $\textsc{Cons}(q)$ and $\textsc{Certainty}(q)$ in place of $\textsc{Cons}(q, \Sigma)$ and $\textsc{Certainty}(q, \Sigma)$, respectively.

Complexity of Consistent Answers: There has been an extensive study of the consistent answers of conjunctive queries [5], [11], [25], [26], [27], [28], [29], [30], [31]. If Σ is a fixed set of key constraints and q is a boolean conjunctive query, then $\textsc{Certainty}(q, \Sigma)$ is always in coNP, but, depending on the query and the constraints, $\textsc{Certainty}(q, \Sigma)$ exhibits a variety of behaviors within coNP. The most definitive result to date is a *trichotomy* theorem by Koutris and Wijsen [30], [31]; it asserts that if q is a self-join-free (no repeated relation symbols) boolean conjunctive query with one key constraint per relation, then $\textsc{Certainty}(q)$ is either SQL-rewritable, or in P but not SQL-rewritable, or coNP-complete. For example, if q is the query $\exists x, y, z(R(\underline{x}, y) \wedge S(\underline{z}, y))$, then $\textsc{Certainty}(q)$ is coNP-complete [25].

Boolean Satisfiability and SAT Solvers: Boolean Satisfiability (SAT) is arguably the prototypical and the most widely studied NP-complete problem. SAT is the following decision problem: *given a boolean formula φ, is φ satisfiable?* Significant progress has been made on developing SAT-solvers, so much so that the advances in this area of research are often referred to as the "SAT Revolution" [32]). Typically, a SAT-solver takes a boolean formula φ in *conjunctive normal form* (CNF) as an input and outputs a satisfying assignment for φ (if one exists) or tells that the formula φ is unsatisfiable.

SAT-solvers are capable of solving quickly SAT-instances with millions of clauses and variables. SAT-solvers have been widely used in both academia and industry as general-purpose tools. Indeed, many real-world problems from a variety of domains, including scheduling, protocol design, software verification, and model checking, can be naturally encoded as SAT-instances, and solved quickly using solvers, such as Glucose [33] and CaDiCaL [34]. Furthermore, SAT-solvers have been used in solving open problems in mathematics [35], [36]. In [16], we used SAT-solvers to build a prototypical system for consistent query answering, which we called CAvSAT. This system can compute consistent answers to unions of SPJ queries over relational databases that are inconsistent w.r.t. a fixed set of arbitrary denial constraints.

III. Range Consistent Answers

Frequently asked database queries often involve one of the standard aggregation operators COUNT(A), COUNT($\star$), SUM(A), AVG(A), MIN(A), MAX(A), and, possibly, a GROUP BY clause. In what follows, we will use the term *aggregation queries* to refer to queries with aggregate operators and with or without a GROUP BY clause. Thus, in full generality, an aggregation query can be expressed as

$Q :=$ SELECT $Z, f(A)$ FROM $T(U, Z, A)$ GROUP BY Z,

where $f(A)$ is one of the aforementioned aggregate operators and $T(U, Z, A)$ is the relation returned by a query q, which typically is a conjunctive query or a union of conjunctive queries expressed in SQL. This query q is called the *underlying query* of Q, the attribute represented by the variable w is called the *aggregation attribute*, and the attributes represented by Z are called the *grouping attributes*. A *scalar aggregation* query is one without a GROUP BY clause.

It is often the case that an aggregation query returns different answers on different repairs of an inconsistent database; thus, even for a scalar aggregation query, there is typically *no* consistent answer as per the definition of consistent answers given earlier. In fact, to produce an empty set of consistent answers, it suffices to have just two repairs on which a scalar aggregation query returns difference answers. To obtain more meaningful answers to aggregation queries, Arenas et al. [2]

proposed the *range consistent answers*, as an alternative notion of consistent answers. For a scalar aggregation query Q, the set of *possible answers* to Q on an inconsistent instance $\mathcal{I}$ consists of the answers to Q over all repairs of $\mathcal{I}$, i.e., $\text{Poss}(Q, \Sigma) = \{Q(\mathcal{J}) \mid \mathcal{J} \text{ is a repair of } \mathcal{I} \text{ w.r.t. } \Sigma\}$. The *range consistent answers* to Q on $\mathcal{I}$ is the interval $[glb(Q, \mathcal{I}), lub(Q, \mathcal{I})]$, where the endpoints of this interval are, respectively, the greatest lower bound (glb) and the least upper bound (lub) of the set $\text{Poss}(Q, \Sigma)$ of possible answers to Q on $\mathcal{I}$.

For example, the range consistent answers of the query

```
SELECT SUM(ACCOUNTS.BAL) FROM ACCOUNTS,
CUSTACC WHERE ACCOUNTS.ACCID = CUSTACC.ACCID
AND CUSTACC.CID = 'C2'
```

on the instance in Table I is the interval $[900, 2200]$. The guarantee is that no matter how the database $\mathcal{I}$ is repaired, the answer to the query is guaranteed to be in the range between 900 and 2200. Note that, the glb-answer comes from a repair of $\mathcal{I}$ that contains the fact f_9, while the lub-answer is from a repair that contains the fact f_8.

Arenas et al. [3] focused on scalar aggregation queries only. Fuxman, Fazli, and Miller [4] extended the notion of range consistent answers to aggregation queries with grouping, i.e., to aggregation queries of the form

$Q :=$ SELECT $Z, f(A)$ FROM $T(U, Z, A)$ GROUP BY Z.

For such queries, a tuple $(T, [glb, lub])$ is a *range consistent answer* to Q on $\mathcal{I}$, if the following conditions hold:

- For every repair $\mathcal{J}$ of $\mathcal{I}$, there exists d s.t. $(T, d) \in Q(J)$ and $glb \leq d \leq lub$.
- For some repair $\mathcal{J}$ of $\mathcal{I}$, we have that $(T, glb) \in Q(J)$
- For some repair $\mathcal{J}$ of $\mathcal{I}$, we have that $(T, lub) \in Q(J)$.

If Q is an aggregation query, $\text{CONS}(Q)$ is the problem: given an instance $\mathcal{I}$, compute the range semantics of Q on $\mathcal{I}$.

Complexity of Range Consistent Answers: Arenas et al. [2] investigated the computational complexity of the range consistent answers for scalar aggregation queries of the form

SELECT $f(A)$ FROM $R(U, A)$,

where $f(A)$ is one of the standard aggregation operators and $R(U, A)$ is a relational schema with functional dependencies. Two relevant findings are as follows.

- If the relational schema $R(U, A)$ has at most one functional dependency and $f(A)$ is one of the aggregation operators MIN(A), MAX(A), SUM(A), COUNT$(*)$, AVG(A), then the range consistent answers of the query SELECT f(A) FROM $R(U, A)$ is in P.
- There is a relational schema $R(U, A)$ with one key dependency s.t. the range consistent answers of the query SELECT COUNT(A) FROM $R(U, A)$ are NP-complete.

It remains an open problem to pinpoint the complexity of the range consistent answers for richer aggregation queries

$Q :=$ SELECT $Z, f(A)$ FROM $T(U, Z, A)$ GROUP BY Z,

where $T(U, Z, A)$ is the relation returned by a conjunctive query q or by a union $q := q_1 \cup \cdots \cup q_k$ of conjunctive queries. It can be shown, however, that if computing the consistent answers $\text{CONS}(q)$ of the underlying query q is a hard problem, then computing the range consistent answers $\text{CONS}(Q)$ of the aggregation query Q is a hard problem as well. This gives rise to the following question: what can we say about the complexity of the range consistent answers $\text{CONS}(Q)$ if computing the consistent answers $\text{CONS}(q)$ of the underlying query is an easy problem?

Fuxman and Miller [25] identified a class, called C_{forest}, of self-join free conjunctive queries whose consistent answers are SQL-rewritable. In his PhD thesis, Fuxman [37] introduced the class $C_{aggforest}$ consisting of all aggregation queries such that the aggregation operator is one of MIN(A), MAX(A), SUM(A), COUNT$(*)$, the underlying query q is a conjunctive query in C_{forest}, and there is one key constraint for each relation in the underlying query q. Fuxman [37] showed that the range consistent answers of every query in $C_{aggforest}$ are SQL-rewritable (earlier, similar results for a proper subclass of $C_{aggforest}$ were obtained by Fuxman, Fazli, and Miller).

It is known that there are self-join free conjunctive queries outside the class C_{forest} whose consistent answers are SQL-rewritable. Koutris and Wijsen [31] characterized the self-join free conjunctive queries whose consistent answers are SQL rewritable. However, the SQL rewritability of aggregation queries beyond those in $C_{aggforest}$ has not been investigated. Here, we show that there exists a self-join-free conjunctive query whose consistent answers are SQL-rewritable, but this property is not preserved when an aggregation operator is added on top of it. For this, we reduce the NP-complete problem MAXIMUM CUT [38] to the problem of computing the range consistent answers to an aggregation query involving SUM and whose underlying conjunctive query has SQL-rewritable consistent answers. The proof of the next result is given in the extended version of this paper on arXiv [20].

Theorem III.1. *Let $\mathcal{R}$ be a relational schema with three relations $R_1(\underline{A_1}, B_1)$, $R_2(\underline{A_2}, B_2)$, and $R_3(\underline{A_1, B_1, A_2, B_2, C})$. Let Q be the aggregation query:*

$Q :=$ *SELECT SUM(A) FROM* $q(A)$,

where $q(A)$ is the following self-join-free conjunctive query:

$\exists x \exists y\ R_1(\underline{x}, \text{'r'}) \wedge R_2(\underline{y}, \text{'b'}) \wedge R_3(\underline{x, \text{'r'}, y, \text{'b'}, A})$.

Then the following two statements hold.

1) $\text{CONS}(q)$ *is SQL-rewritable.*
2) $\text{CONS}(Q)$ *is NP-hard.*

IV. Consistent Answers via SAT Solving

Here, we give polynomial-time reductions from computing the range consistent answers of aggregation queries with the operators COUNT$(*)$, COUNT(A), SUM(A) to variants of SAT. In the extended version of this paper on arXiv [20], we give reductions of the range consistent answers of aggregation queries with the operators MIN(A), MAX(A) to iterative SAT. The reductions in this section assume that the database schema has one key constraint per relation; in Section V, we show how these reductions can be extended to schemata with arbitrary denial constraints. Our reductions rely on several well known *optimization* variants of SAT that we describe next.

- Weighted MaxSAT (or WMaxSAT) is the maximization variant of SAT in which each clause is assigned a positive weight and the goal is to find an assignment that maximizes

the sum of the weights of the satisfied clauses. We write $(l_1 \vee \cdots \vee l_k, w)$ to denote a clause $(l_1 \vee \cdots \vee l_k)$ with weight w.

- Partial MaxSAT (or PMaxSAT) is the maximization variant of SAT in which some clauses of the formula are assigned infinite weight (*hard clauses*), while each of the rest is assigned weight one (*soft clauses*). The goal is to find an assignment that satisfies all hard clauses and the maximum number of soft clauses. If the hard clauses of a PMaxSAT instance are not simultaneously satisfiable, then we say that the instance is *unsatisfiable*. For simplicity, a hard clause $(l_1 \vee \cdots \vee l_k, \infty)$ is denoted as $(l_1 \vee \cdots \vee l_k)$.
- Weighted Partial MaxSAT (or WPMaxSAT) is the maximization variant of SAT where some of the clauses of the formula are assigned infinite weight (*hard clauses*), while each of the rest is assigned a positive weight (*soft clauses*). The goal is to find an assignment that satisfies all hard clauses and maximizes the sum of weights of the satisfied soft clauses.

Modern solvers, such as MaxHS [18], can efficiently solve large instances of these maximization variants of SAT. Note that these maximization problems have dual minimization problems, called WMinSAT, PMinSAT, and WPMinSAT, respectively. For example, in WPMinSAT, the goal is to find an assignment that satisfies all hard clauses and minimizes the sum of weights of the satisfied soft clauses. These minimization problems are of interest to us, because some of the computations of the range consistent answers have natural reductions to such minimization problems. At present, the only existing WPMinSAT solver is MinSatz [39]. Since this solver has certain size limitations, we will deploy Kügel's technique [40] to first reduce WPMinSAT to WPMaxSAT, and then use the MaxHS solver in our experiments. This technique uses the concept of CNF-*negation* – see [40], [41].

We also need to consider the notions of *key-equal groups* of facts and the *bag of witnesses* to queries. Let $\mathcal{I}$ be a database instance and let $vals(\mathcal{I})$ be the set of values occurring in $\mathcal{I}$.

- We say that two facts of a relation R of $\mathcal{I}$ are *key-equal*, if they agree on the key attributes of R. A set S of facts of $\mathcal{I}$ is called a *key-equal group* of facts if every two facts in S are key-equal, and no fact in S is key-equal to some fact in $\mathcal{I} \backslash S$.
- Let $q(\mathbf{z}) := \exists \mathbf{w}\ (R_1(\mathbf{x}_1) \wedge ... \wedge R_m(\mathbf{x}_m))$ be a conjunctive query, where each $\mathbf{x}_i$ is a tuple of variables and constants, and let $\mathbf{a} \in q(\mathcal{I})$ be an answer to q on $\mathcal{I}$. Let *vars*(q) and *cons*(q) be the sets of variables and constants occurring in q. A function $f : vars(q) \cup cons(q) \rightarrow vals(\mathcal{I})$ is a *witnessing assignment to* $\mathbf{a}$ if the following hold: $f(\mathbf{z}) = \mathbf{a}$; if x_j is a constant in q, then $f(x_j) = x_j$; and if $R_i(x_1, \cdots, x_n)$ is an atom of q, then $R_i(f(x_1), \cdots, f(x_n))$ is a fact of $\mathcal{I}$. We say that a set S of facts from $\mathcal{I}$ is a *witness to* $\mathbf{a}$ if there is a witnessing assignment f to $\mathbf{a}$ such that $S = \{R_i(f(x_1), \cdots, f(x_n)) : R_i(x_1, \cdots, x_n)$ is an atom of $q\}$.

Note that two distinct witnessing assignments to an answer may give rise to the same witness. Thus, we consider the *bag of witnesses to an answer*, i.e., the bag consisting of witnesses arising from all witnessing assignments to that answer, where each witness S is accompanied by its multiplicity, an integer denoting the number of witnessing assignments that gave rise to S. Finally, we define the *bag of witnesses to a conjunctive query* as the bag union of the bags of witnesses over all answers to q on $\mathcal{I}$ (in the bag union the multiplicities of the same set are added). The *bag of witnesses to a union* $q := q_1 \cup \cdots \cup q_k$ of conjunctive queries is the bag union of the bags of witnesses to each conjunctive query q_i in q. The bag of witnesses will be used in computing the range consistent answers to aggregation queries. In effect, the bag of witnesses corresponds to the provenance polynomials of conjunctive queries and their unions [42], [43].

It is easy to verify that both the key equal groups and the bag of the witnesses can be computed using SQL queries.

A. Answering Queries without Grouping

Let $\mathcal{R}$ be a database schema with one key constraint per relation, and let Q be the scalar aggregation query `SELECT` f `FROM` $T(U, A)$, where f is one of the operators `COUNT(*)`, `COUNT(`A`)`, `SUM(`A`)`, and $T(U, A)$ is a union of conjunctive queries over $\mathcal{R}$.

We will reduce the range consistent answers $\textsc{Cons}(Q)$ of Q to PMaxSAT and to WPMaxSAT. We first give the intuition behind Reduction IV.1. We represent each fact of the database instance with a boolean variable, which allows us to encode the inconsistencies in the database into α-clauses (Step 1), and the answers to the input query into β-clauses and γ-clauses of the CNF formula ϕ (Steps 2a and 2b). Reduction IV.1 makes sure that every repair of the database instance uniquely corresponds to an assignment to ϕ. Importantly, the repairs that contain the *glb*-answer and the *lub*-answer to the query correspond to the optimal assignments to ϕ, namely, maximum satisfying assignment and the minimum satisfying assignment respectively (Proposition IV.1).

Reduction IV.1. *Let $Q :=$ `SELECT` f `FROM` $T(U, A)$ be an aggregation query, where f is one of the operators `COUNT(*)`, `COUNT(`A`)`, and `SUM(`A`)`. Let $\mathcal{I}$ be an $\mathcal{R}$-instance and $\mathcal{G}$ be the set of key-equal groups of facts of $\mathcal{I}$. For each fact f_i of $\mathcal{I}$, introduce a boolean variable x_i. Let $\mathcal{W}$ be the bag of witnesses to the query q^* on $\mathcal{I}$, where*

$$q^* := \begin{cases} \exists U \exists A\ T(U, A) & \text{if } f \text{ is } \texttt{COUNT(*)} \\ \exists U\ T(U, A) & \text{if } f \text{ is } \texttt{COUNT(}A\texttt{)} \text{ or } \texttt{SUM(}A\texttt{)}. \end{cases}$$

Construct a partial CNF-formula ϕ (if f is `COUNT(*)` or `COUNT(`A`)`) or a weighted partial CNF-formula ϕ (if f is `SUM(`A`)`) as follows:

(1) For each $G_j \in \mathcal{G}$,

- *construct a hard clause $\alpha_j = \bigvee_{f_i \in G_j} x_i$.*
- *for each pair (f_m, f_n) of facts in G_j such that $m \neq n$, construct a hard clause $\alpha_j^{mn} = (\neg x_m \vee \neg x_n)$.*

(2a) If f is `COUNT(*)` or `COUNT(`A`)`, then for each witness $W_j \in \mathcal{W}$, construct a soft clause $\beta_j = \big(\bigvee_{f_i \in W_j} \neg x_i, m_j\big)$, where m_j is the multiplicity of W_j in $\mathcal{W}$.

Construct a partial CNF-instance

$$\phi = \Big(\bigwedge_{j=1}^{|\mathcal{G}|} \alpha_j\Big) \wedge \Big(\bigwedge_{j=1}^{|\mathcal{G}|} \Big(\bigwedge\nolimits_{\substack{f_m \in \mathcal{G}_j \\ f_n \in \mathcal{G}_j}} \alpha_j^{mn}\Big)\Big) \wedge \Big(\bigwedge_{j=1}^{|\mathcal{W}|} \beta_j\Big).$$

(2b) If f is `SUM(A)`*, let $\mathcal{W}_P$ and $\mathcal{W}_N$ be the subsets of $\mathcal{W}$ such that for each $W_j \in \mathcal{W}$, we have $W_j \in \mathcal{W}_P$ iff $q^*(W_j) > 0$, and $W_j \in \mathcal{W}_N$ iff $q^*(W_j) < 0$. Let also $w_j = m_j * ||q^*(W_j)||$, where $||q^*(W_j)||$ is the absolute value of $q^*(W_j)$. Construct a weighted soft clause β_j and a conjunction γ_j of hard clauses as follows. If $W_j \in \mathcal{W}_\mathcal{N}$, introduce a new variable y_j and let $\beta_j = (y_j, w_j)$ and*

$$\gamma_j = \Big(\Big(\bigvee_{f_i \in W_j} \neg x_i\Big) \vee y_j\Big) \wedge \Big(\bigwedge_{f_i \in W_j} (\neg y_j \vee x_i)\Big);$$

else, let $\beta_j = \Big(\bigvee_{f_i \in W_j} \neg x_i, w_j\Big)$ and do not construct γ_j. Construct a weighted partial CNF-instance

$$\phi = \Big(\bigwedge_{j=1}^{|\mathcal{G}|} \alpha_j\Big) \wedge \Big(\bigwedge_{j=1}^{|\mathcal{G}|} \Big(\wedge_{\substack{f_m \in \mathcal{G}_j \\ f_n \in \mathcal{G}_j}} \alpha_j^{mn}\Big)\Big) \wedge \Big(\bigwedge_{j=1}^{|\mathcal{W}|} \beta_j\Big) \wedge \Big(\bigwedge_{W_j \in \mathcal{W}_\mathcal{N}} \gamma_j\Big)$$

Purpose of the components of ϕ in Reduction IV.1

- Each α_j-clause encodes the *"at-least-one"* constraint for each key-equal group G_j in the sense that satisfying α_j requires setting at least one variable corresponding to a fact in G_j to true. Similarly, each $\alpha_j^m n$-clause encodes the *"at-most-one"* constraint for G_j. In effect, every assignment that satisfies all α-clauses sets exactly one variable corresponding to the facts from each key-equal group to true, and thus uniquely corresponds to a repair of $\mathcal{I}$.
- Satisfying a β_j-clause constructed in Step 2a requires setting at least one variable corresponding to the facts of a witness W_j to q^* on $\mathcal{I}$ to false. Thus, if s is an assignment that satisfies all α-clauses, then β_j is satisfied by s if and only if $W_j \notin \mathcal{J}$, where $\mathcal{J}$ is a repair corresponding to s.
- The β_j-clauses constructed in Step 2b serve the same purpose as the ones from Step 2a, but here they are constructed only for the witnesses in $\mathcal{W}_P$. For the witnesses in $\mathcal{W}_N$, the β_j-clauses encode the condition that β_j is satisfied if and only if all variables corresponding to the facts in W_j are set to true. The hard γ_j-clauses are used solely to express the equivalence $y_j \leftrightarrow (\bigwedge_{f_i \in W_j} x_i)$ in conjunctive normal form.

The number of α-clauses is $O(n)$, where n is the size of the database; the number of β-clauses and γ-clauses combined is $O(n^k)$, where k is the number of relation symbols in Q.

Proposition IV.1. *Let $Q :=$* `SELECT` *f* `FROM` *$T(U, A)$ be an aggregation query, where f is one of the operators* `COUNT(*)`*,* `COUNT(A)`*, and* `SUM(A)`*. In a maximum (a minimum) satisfying assignment of the WPMaxSAT-instance ϕ constructed using Reduction IV.1, the sum of weights of the falsified clauses is the glb-answer (lub-answer) in* $\textsc{Cons}(Q)$ *on $\mathcal{I}$.*

Example IV.1. *Let $\mathcal{I}$ be a database instance from Table I, and Q be the following aggregation query which counts the number of customers who have an account in their own city:*

```
SELECT COUNT(*) FROM CUST, ACC, CUSTACC
WHERE CUST.CID = CUSTACC.CID AND ACC.ACCID
= CUSTACC.ACCID AND CUST.CITY = ACC.CITY
```

From Reduction IV.1, we construct the following clauses:
α-clauses: $x_1, (x_2 \vee x_3), x_4, x_5, x_6, x_7, (x_8 \vee x_9), x_{10}$*;*
α^{mn}-clauses: $(\neg x_2 \vee \neg x_3), (\neg x_8 \vee \neg x_9)$*;*
β-clauses: $(\neg x_1 \vee \neg x_6, 1), (\neg x_2 \vee \neg x_7, 1), (\neg x_3 \vee \neg x_9, 1)$.
By Proposition IV.1, we have that $\textsc{Cons}(Q, \mathcal{I}) = [1, 2]$.

Example IV.2. *Let us again consider the database instance $\mathcal{I}$ from Table I, and the following aggregation query Q:*

```
SELECT SUM(ACC.BAL) FROM CUST, ACC, CUSTACC
WHERE CUST.CID = CUSTACC.CID AND ACC.ACCID =
CUSTACC.ACCID AND CUST.CNAME = 'Mary'
```

The hard clauses constructed using Reduction IV.1 are same as the ones from Example IV.1. The rest of the clauses are:
β-clauses: $(\neg x_2 \vee \neg x_7, 1000)$, $(\neg x_3 \vee \neg x_7, 1000)$, $(\neg x_2 \vee \neg x_8, 1200)$, $(\neg x_3 \vee \neg x_8, 1200)$, $(y_1, 100)$, $(y_2, 100)$.
γ-clauses: $(\neg x_2 \vee \neg x_9 \vee y_1)$, $(\neg y_1 \vee x_2)$, $(\neg y_1 \vee x_9)$, $(\neg x_3 \vee \neg x_9 \vee y_2)$, $(\neg y_2 \vee x_3)$, $(\neg y_2 \vee x_9)$.
By Proposition IV.1, we have that $\textsc{Cons}(Q, \mathcal{I}) = [900, 2200]$.

B. Handling the `DISTINCT` *Keyword*

Let $Q :=$ `SELECT` f `FROM` $T(U, A)$ be an aggregation query, where f is either `COUNT(DISTINCT` A`)` or `SUM(DISTINCT` A`)`. Solving the PMaxSAT or the WPMaxSAT instance constructed using Reduction IV.1 may yield incorrect glb and lub answers to Q, if the database contains multiple witnesses with the same value for attribute A. For example, consider the database instance $\mathcal{I}$ from Table I, and a query Q:

```
SELECT COUNT(DISTINCT ACC.TYPE) FROM ACC.
```

The correct glb and lub-answers in $\textsc{Cons}(Q, \mathcal{I})$ are both 2, but solutions to the PMaxSAT and PMinSAT instances constructed using Reduction IV.1 yield both answers as 4. The reason behind this is that the soft clauses $\neg x_6$ and $\neg x_7$ both correspond to the account type Checking, and similarly $\neg x_8$, $\neg x_9$, and $\neg x_{10}$ all correspond to the account type Saving. The hard clauses in the formula ensure that x_6, x_7, x_{10}, and one of x_8 and x_9 are true, thus counting both Checking and Saving account types exactly twice in every satisfying assignment to the formula. This can be handled by modifying the β-clauses in Reduction IV.1 as follows.

Let $\mathcal{A}$ denote a set of distinct answers to the query $q^*(A) := \exists U\, T(U, A)$. For each answer $b \in \mathcal{A}$, let $\mathcal{W}^b$ denote a subset of $\mathcal{W}$ such that for every minimal witness $W \in \mathcal{W}^b$, we have that $q^*(W) = b$. The idea is to use auxiliary variables to construct one soft clause for every distinct answer $b \in \mathcal{A}$, such that it is true if and only if no witness in $\mathcal{W}^b$ is present in a repair corresponding to the satisfying assignment. First, for every witness $W_j^b \in \mathcal{W}^b$, we introduce an auxiliary variable z_j^b that is true if and only if W_j^b is not present in the repair. Then, we introduce an auxiliary variable v^b which is true if and only if all z^b-variables are true. These constraints are encoded in the set H^b returned by Algorithm 1, and are forced by making clauses in H^b hard. For every answer $b \in \mathcal{A}$, Algorithm 1 also returns one β^b-clause, which serves the same purpose as the β-clauses in Reduction IV.1. Now, a PMaxSAT or a WPMaxSAT instance can be constructed by taking in conjunction all α-clauses from the key-equal groups, the hard γ-clauses if any,

the hard clauses from all H^b-sets, and all soft β^b-clauses. With this, it is easy to see that a maximum (*or minimum*) satisfying assignment to PMaxSAT or WPMaxSAT instance give us the *glb*-answer (*or lub-answer*) in $\text{CONS}(Q)$ – see Example IV.3.

Algorithm 1 Handling DISTINCT

```
1: procedure HANDLEDISTINCT(W^b)
2:     let H^b = ∅                                   //Empty set of clauses
3:     for W_j^b ∈ W^b do
4:         H^b = H^b ⋃ { ( ¬z_j^b ∨ ( ⋁_{f_i ∈ W_j^b} ¬x_i ) ) }
5:         for f_i ∈ W_j^b do
6:             H^b = H^b ⋃ {(z_j^b ∨ x_i)}
7:     H^b = H^b ⋃ { ( ¬v^b ∨ ( ⋁_{W_j^b ∈ W^b} ¬z_j^b ) ) }
8:     for W_j^b ∈ W^b do
9:         H^b = H^b ⋃ {(¬v^b ∨ z_j^b)}
10:    let β^b = (v^b, 1)
11:    if (f is SUM(DISTINCT A)) then
12:        β^b = (v^b, ||b||)
13:        if b < 0 then β^b = (¬v^b, ||b||)
14:    return H^b, β^b
```

Example IV.3. *Consider the following aggregation query Q on the database instance $\mathcal{I}$ from Table I:*

```
SELECT COUNT(DISTINCT ACC.TYPE) FROM ACC
```

We have that $\mathcal{A} = \{$'Checking', 'Saving'$\}$. Let us denote these two answers by a_1 and a_2 respectively. Since every witness to the query consists of a single fact, every y^a-variable is equivalent to a single literal, for example, $y_1^{a_1} \leftrightarrow \neg x_6$ and $y_2^{a_1} \leftrightarrow \neg x_7$. As a result, it is unnecessary to introduce any y^a-variables at all. Thus, we construct the following clauses from Reduction IV.1 and Algorithm 1:

α-clauses: $x_6, x_7, (x_8 \vee x_9), x_{10}$; α^{mn}-clauses: $(\neg x_8 \vee \neg x_9)$; $H^{a_1} : (x_6 \vee x_7 \vee v^{a_1}), (\neg v^{a_1} \vee \neg x_6), (\neg v^{a_1} \vee \neg x_7)$; $H^{a_2} : (x_8 \vee x_9 \vee x_{10} \vee v^{a_2}), (\neg v^{a_2} \vee \neg x_8), (\neg v^{a_2} \vee \neg x_9), (\neg v^{a_2} \vee \neg x_{10})$; *$\beta$-clauses: $(v^{a_1}, 1), (v^{a_2}, 1)$.*

The maximum and minimum satisfying assignments to the PMaxSAT and PMinSAT instances constructed using these clauses falsify both β-clauses, since $\text{CONS}(Q, \mathcal{I}) = [2, 2]$.

C. Answering Queries with Grouping

Let Q be the aggregation query

SELECT Z, f FROM $T(U, Z, w)$ GROUP BY Z,

where f is one of COUNT$(*)$, COUNT(A), SUM(A), MIN(A), or MAX(A), and $T(U, A)$ is a relation expressed by a union of conjunctive queries on $\mathcal{R}$. We refer to the attributes in Z as the grouping attributes. For aggregation queries with grouping, it does not seem feasible to reduce $\text{CONS}(Q)$ to a single PMaxSAT or a WPMaxSAT instance because for each group of consistent answers, the *glb*-answer and the *lub*-answer may realize in different repairs of the inconsistent database. To illustrate this, consider the database from Table I and a query

$Q :=$ SELECT COUNT(*) FROM CUST GROUP BY CITY.

Notice that, the *glb*-answers (LA, 2) and (SF, 1) in $\text{CONS}(Q)$ come from two different repairs of relation CUST, namely, $\{f_1, f_3, f_4, f_5\}$ and $\{f_1, f_2, f_4, f_5\}$ respectively. However, the reductions from the preceding section can be used to compute the bounds to each consistent group of answers independently. For a given aggregation query Q with grouping, we first compute the consistent answers to an underlying conjunctive query $q(Z) := \exists U, A\, T(U, Z, A)$. Then, for each answer b in $\text{CONS}(q)$, we compute the *glb* and *lub*-answers to the query $Q' :=$ SELECT f FROM $T(U, Z, A) \wedge (Z = b)$ via PMaxSAT or WPMaxSAT solving as shown in Algorithm 2.

Algorithm 2 Consistent Answers to Queries With Grouping

Let $\mathcal{I}$ be an inconsistent database instance, and Q be an aggregation query of the form
$Q :=$ SELECT Z, f FROM $T(U, Z, A)$ GROUP BY Z.

```
1: procedure CONSAGGGROUPING(Q)
2:     let Ans = ∅
3:     let q(Z) := ∃U, A T(U, Z, A)
4:     let A_c = CONS(q, I)
5:     for b ∈ A_c do
6:         let Q' := SELECT f FROM T(U, Z, A) ∧ (Z = b)
7:         let [GLB_A, LUB_A] = CONS(Q', I)
8:         A_ans = A_ans ∪ (b, [GLB_A, LUB_A])
9:     return A_ans
```

As noted earlier, the bags of witnesses used in the preceding reductions capture the provenance of unions of conjunctive queries in the provenance polynomials model of [42], [43]. In [44], it was shown that a stronger provenance model is needed to express the provenance of aggregation queries, a model that uses a tensor product combining annotations with values. A future direction of research is to investigate whether this stronger provenance model can be used to produce more direct reductions of the range consistent answers to SAT.

V. Beyond Key Constraints

Key constraints and functional dependencies are important special cases of *denial constraints* (DCs), which are expressible by first-order formulas of the form $\forall x_1, ..., x_n \neg(\varphi(x_1, ..., x_n) \wedge \psi(x_1, ..., x_n))$, or, equivalently, $\forall x_1, ..., x_n (\varphi(x_1, ..., x_n) \rightarrow \neg\psi(x_1, ..., x_n))$, where $\varphi(x_1, ..., x_n)$ is a conjunction of atomic formulas and $\psi(x_1, ..., x_n)$ is a conjunction of expressions of the form $(x_i \text{ op } x_j)$ with each op a built-in predicate, such as $=, \neq, <, >, \leq, \geq$. In words, a denial constraint prohibits a set of tuples that satisfy certain conditions from appearing together in a database instance. If Σ is a fixed finite set of denial constraints and Q is an aggregation query without grouping, then the following problem is in coNP: given a database instance $\mathcal{I}$ and a number t, is t the *lub*-answer *(or the glb-answer)* in $\text{CONS}(Q, \mathcal{I})$ w.r.t. Σ? This is so because to check that t is not the *lub*-answer *(or the glb-answer)*, we guess a repairs $\mathcal{J}$ of $\mathcal{I}$ and verify that $t > Q(J)$ *(or* $t > Q(J)$*)*. In all preceding reductions, the α-clauses capture

the inconsistency in the database arisen due to the key violations to enforce every satisfying assignment to uniquely correspond to a repair of the initial inconsistent database instance. Importantly, the α-clauses are independent of the input query. In what follows, we provide a way to construct clauses to capture the inconsistency arising due to the violations of denial constraints. Thus, replacing the α-clauses in the reductions from Section IV by the ones provided below allows us to compute consistent answers over databases with a fixed finite set of arbitrary denial constraints. The reduction relies on the notions of *minimal violations* and *near-violations* to the set of denial constraints that we introduce next.

Assume that Σ is a set of denial constraints, $\mathcal{I}$ is an $\mathcal{R}$-instance, and S is a sub-instance of $\mathcal{I}$.

- We say that S is a *minimal violation* to Σ, if $S \not\models \Sigma$ and for every set $S' \subset S$, we have that $S' \models \Sigma$.
- Let f be a fact of $\mathcal{I}$. We say that S is a *near-violation* w.r.t. Σ and f if $S \models \Sigma$ and $S \cup \{f\}$ is a minimal violation to Σ. As a special case, if $\{f\}$ itself is a minimal violation to Σ, we say that there is exactly one near-violation w.r.t. f, and it is the singleton $\{f_{true}\}$, where f_{true} is an auxiliary fact.

Let $\mathcal{R}$ be a database schema, Σ be a fixed finite set of denial constraints on $\mathcal{R}$, Q be an aggregation query without grouping, and $\mathcal{I}$ be an $\mathcal{R}$-instance.

Reduction V.1. *Given an $\mathcal{R}$-instance $\mathcal{I}$, compute $\mathcal{V}$, the set of minimal violations to Σ on $\mathcal{I}$, and $\mathcal{N}^i$, the set of near-violations to Σ, on $\mathcal{I}$, w.r.t. each fact $f_i \in \mathcal{I}$. For each fact f_i of $\mathcal{I}$, introduce a boolean variable x_i. For the auxiliary fact f_{true}, introduce a constant $x_{true} = true$, and for each $N^i_j \in \mathcal{N}^i$, introduce a boolean variable p^i_j.*

1) For each $V_j \in \mathcal{V}$, construct a clause $\alpha_j = \bigvee_{f_i \in V_j} \neg x_i$.

2) For each $f_i \in I$, construct a clause $\gamma_i = x_i \vee \Big(\bigvee_{N^i_j \in \mathcal{N}^i} p^i_j \Big)$.

3) For each p^i_j, construct an expression $\theta^i_j = p^i_j \leftrightarrow \bigwedge_{f_d \in N^i_j} x_d$.

4) Construct the following boolean formula ϕ:

$$\phi = \Big(\bigwedge_{i=1}^{|\mathcal{V}|} \alpha_i \Big) \wedge \Big(\bigwedge_{i=1}^{|\mathcal{I}|} \Big(\Big(\bigwedge_{j=1}^{|\mathcal{N}^i|} \theta^i_j \Big) \wedge \gamma_i \Big) \Big)$$

Proposition V.1. *The boolean formula ϕ constructed using Reduction V.1 can be transformed to an equivalent CNF-formula ϕ whose size is polynomial in the size of $\mathcal{I}$. The satisfying assignments to ϕ and the repairs of $\mathcal{I}$ w.r.t. Σ are in one-to-one correspondence.*

Proof. (*Sketch*) The first part is proved using basic rules of propositional logic. For the second part, consider a satisfying assignment s to ϕ and construct a database instance $\mathcal{J}$ such that $f_i \in \mathcal{J}$ if and only if $s(x_i) = 1$. The α-clauses assert that no minimal violation to Σ is present in $\mathcal{J}$, i.e., $\mathcal{J}$ is a consistent subset of $\mathcal{I}$. The γ-clauses and the θ-expressions encode the condition that, for every fact $f \in \mathcal{I}$, either $f \in \mathcal{J}$ or at least one near-violation w.r.t. Σ and f is in $\mathcal{J}$, making sure that $\mathcal{J}$ is indeed a repair of $\mathcal{I}$. In the other direction, one can construct a satisfying assignment s to ϕ from a repair $\mathcal{J}$ of $\mathcal{I}$ by setting $s(x_i) = 1$ if and only if $f_i \in \mathcal{J}$. □

VI. Experimental Evaluation

We evaluate the performance of AggCAvSAT over both synthetic and real-world databases. The first set of experiments includes a comparison of AggCAvSAT with an existing SQL-rewriting-based CQA system, namely, ConQuer, over synthetically generated TPC-H databases having one key constraint per relation. This set of experiments is divided into two parts, based on the method used to generate the inconsistent database instances. In the first part, we use the `DBGen` tool from TPC-H and artificially inject inconsistencies in the generated data; in the second part, we employ the PDBench inconsistent database generator from MayBMS [19] (see Section VI-A1 for details). Next, we assess the scalability of AggCAvSAT by varying the size of the database and the amount of inconsistency present in it. Lastly, to evaluate the performance of the reductions from Section V, we use a real-world Medigap [45] dataset that has three functional dependencies and one denial constraint. All experiments were carried out on a machine running on Intel Core i7 2.7 GHz, 64 bit Ubuntu 16.04, with 8GB of RAM. We used Microsoft SQL Server 2017 as an underlying DBMS, and MaxHS v3.2 solver [18] for solving the WPMaxSAT instances. The AggCAvSAT system is implemented in Java 9.04 and its code is open-sourced at a GitHub repository https://github.com/uccross/cavsat via a BSD-style license. Various features of AggCAvSAT, including its graphical user interface, are presented in a short paper in the demonstration track of the 2021 SIGMOD conference [46].

A. Experiments with TPC-H Data and Queries

1) Datasets: For the first part of the experiments, the data is generated using the `DBGen` data generation tool from the TPC-H Consortium. The TPC-H schema comes with exactly one key constraint per relation, which was ideal for comparing AggCAvSAT against ConQuer [4], [25] (the only existing system for computing the range consistent answers to aggregation queries), because ConQuer does not support more than one key constraint per relation or classes of integrity constraints broader than keys. The `DBGen` tool generates consistent data, so we artificially injected inconsistency by updating the key attributes of randomly selected tuples from the data with the values taken from existing tuples of the same relation. The sizes of the key-equal groups that violate the key constraints were uniformly distributed between two and seven. The database instances were generated in such a way that every repair had the specified size. We experimented with varied degrees of inconsistency, ranging from 5% up to 35% of the tuples violating a key constraint, and with a variety of repair sizes, starting from 500 MB (4.3 million tuples) up to 5 GB (44 million tuples). For the second part, we employed the PDBench database generator from MayBMS [19] to generate four inconsistent database instances with varying degrees of inconsistency (see Table II). In all four instances, the data is generated in such a way that every repair is of size 1 GB.

2) Queries: The standard TPC-H specification comes with 22 queries (constructed using the `QGen` tool). Here, we focus on queries 1, 3, 4, 5, 6, 10, 12, 14, and 19; the other 13

Table II: Inconsistent TPC-H instances using PDBench

	Inconsistency			
Table	**Inst. 1**	**Inst. 2**	**Inst. 3**	**Inst. 4**
CUSTOMER	4.42%	8.5%	16.14%	29.49%
LINEITEM	6.36%	12.09%	22.53%	39.82%
NATION	7.69%	0%	7.69%	7.69%
ORDERS	3.51%	6.77%	12.87%	23.9%
PART	4.93%	9.33%	17.66%	32.16%
PARTSUPP	1.53%	2.96%	5.77%	11.13%
REGION	0%	0%	0%	0%
SUPPLIER	3.69%	7.44%	14.11%	26.51%
Overall	5.36%	10.25%	19.29%	34.72%
	Database size and Repair size (in GB)			
	1.04 & 1.00	1.07 & 1.01	1.14 & 1.02	1.3 & 1.02
	Size of the Largest Key-equal Groups			
	8 tuples	16 tuples	16 tuples	32 tuples

queries have features such as nested subqueries, left outer joins, and negation that are beyond the aggregation queries defined in Section III. In Section VI-A3, we describe our results for queries without grouping. Since six out of the nine queries contained the `GROUP BY` clause, we removed it and added appropriate conditions in the `WHERE` clause based on the original grouping attributes to obtain queries without grouping. We refer to these queries as $Q'_1, Q'_3, \cdots, Q'_{19}$.

3) Results on Queries without Grouping: In the first set of experiments, we computed the range consistent answers of the TPC-H-inspired aggregation queries without grouping via WPMaxSAT solving over a database instance with 10% inconsistency and having repairs of size 1 GB (8 million tuples). Figure 1 shows that much of the evaluation time used by AggCAvSAT is consumed in encoding the CQA instance into a WPMaxSAT instance, while the solver comparatively takes less time to compute the optimal solution. Note that, Q'_5 is not in the class $C_{aggforest}$ and thus ConQuer cannot compute its range consistent answers. AggCAvSAT performs better than ConQuer on seven out of the remaining eight queries.

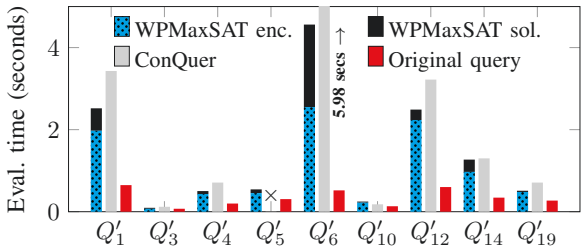

Figure 1: AggCAvSAT vs. ConQuer on TPC-H data generated using the `DBGen`-based tool (10% inconsistency, 1 GB repairs)

Next, we compared the performance of AggCAvSAT and ConQuer on database instances generated using PDBench. Figure 2 shows that AggCAvSAT performs better than ConQuer on PDBench instances with low inconsistency. As the inconsistency increases, the WPMaxSAT solver requires considerably long time to compute the optimal solutions (especially for Q'_6, Q'_{12}, and Q'_{14}). One reason is that the sizes of key-equal groups in PDBench instances with higher inconsistency percentage are large, which translates into clauses of large sizes in the WPMaxSAT instances, hence the solver works hard to solve them. Also, Kügel's reduction [40] from WPMinSAT to WPMaxSAT significantly increases the size of the CNF formula, resulting in higher time for the *lub*-answers.

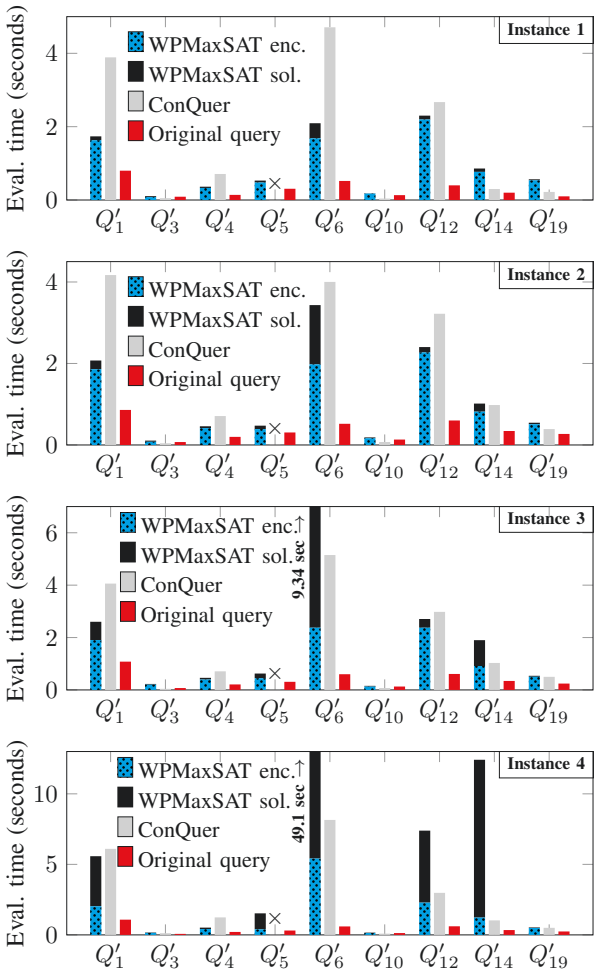

Figure 2: AggCAvSAT vs. ConQuer on PDBench instances

Table III: Average size of CNF formulas for Q'_1, Q'_6, and Q'_{14}

	5%	15%	25%	35%
Q'_1	10.2	34.3	60.6	95.3
Q'_6	28.4	96.2	175.0	271.2
Q'_{14}	6.4	21.1	40.6	62.3

(a) # of variables (in thousands)

	5%	15%	25%	35%
Q'_1	27.6	92.2	163.6	258.1
Q'_6	76.8	259.9	472.9	734.0
Q'_{14}	15.6	51.9	101.0	156.6

(b) # of clauses (in thousands)

	1 GB	3 GB	5 GB
Q'_1	21.3	44.1	105.6
Q'_6	60.9	127.13	304.4
Q'_{14}	13.9	32.9	67.7

(c) # of variables (in thousands)

	1 GB	3 GB	5 GB
Q'_1	57.7	104.1	258.8
Q'_6	165.3	300.7	823.1
Q'_{14}	34.0	73.7	166.6

(d) # of clauses (in thousands)

Next, we varied the inconsistency in the database instances created using the `DBGen`-based data generator while keeping the size of the database repairs constant (1 GB). Figure 3

shows that the evaluation time of AggCAvSAT stays well under ten seconds (except for Q'_6), even if there is more inconsistency in the data. Tables IIIa and IIIb show the average size of the CNF formulas for the top three queries that exhibited the largest CNF formulas. The size of the formulas grows nearly linearly as the inconsistency present in the data grows. The CNF formulas for Q'_6 are significantly larger than the ones corresponding to the other queries since Q'_6 has high selectivity and it is posed against the single largest relation `LINEITEM` which has over 8.2 million tuples in an instance with 35% inconsistency. This also explains why AggCAvSAT takes more time for computing its range consistent answers (Figure 3). In database instances with low inconsistency, the consistent answers to the queries having low selectivity (e.g., Q'_3, Q'_{10}) are sometimes contained in the consistent part of the data, and AggCAvSAT does not need to construct a WPMaxSAT instance at all.

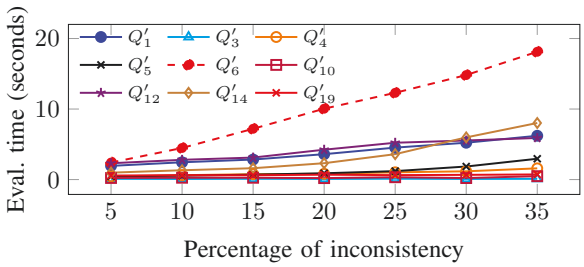

Figure 3: AggCAvSAT on TPC-H data generated using the `DBGen`-based tool (varying inconsistency, 1 GB repairs)

We then evaluated AggCAvSAT's scalability by increasing the sizes of the databases while keeping the inconsistency to a constant 10%. Figure 4 shows that the evaluation time of AggCAvSAT for queries Q'_1, Q'_6, and Q'_{12} increases faster than that for the other queries. This is because the queries Q'_1 and Q'_6 are posed against `LINEITEM` while Q'_{12} involves a join between `LINEITEM` and `ORDERS` resulting in AggCAvSAT spending more time on computing the minimal witnesses to these queries as the size of the database grows. Table IIIc and IIId show that the size of the CNF formulas grows almost linearly w.r.t. the size of the database. The largest CNF formula consisted of over 304000 variables and 823000 clauses and was exhibited by Q'_6 on a database of size 5 GB (47 million tuples). The low selectivity of queries Q'_3, Q'_{10}, and Q'_{19} resulted in very small CNF formulas, even on large databases.

4) Results on Queries with Grouping: In this set of experiments, we focus on TPC-H queries 1, 3, 4, 5, 10, and 12, as the queries 6, 14, and 19 did not contain grouping. We evaluated the performance of AggCAvSAT and compared it to ConQuer on a database with 10% inconsistency w.r.t. primary keys (Figure 5). The repairs are of size 1 GB. AggCAvSAT computes the consistent answers to the underlying conjunctive query using the reductions from [16] which are, precisely, the consistent groups in the range consistent answers to the aggregation query. For each of these groups, it computes the *glb*-answer and the *lub*-answer using reductions to WPMaxSAT.

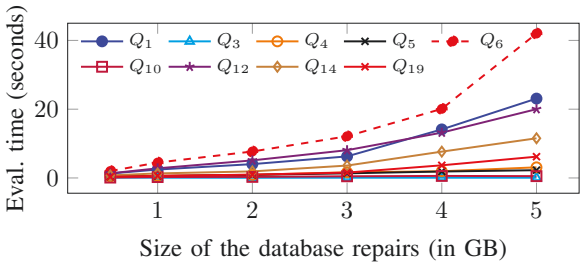

Figure 4: AggCAvSAT on TPC-H data generated using the `DBGen`-based tool (varying database sizes, 10% inconsistency)

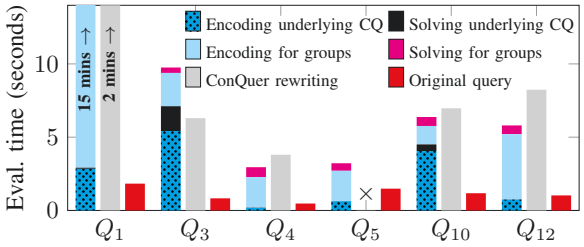

Figure 5: AggCAvSAT vs. ConQuer on TPC-H data generated using the `DBGen`-based tool (10% inconsistency, 1 GB repairs)

The overhead of computing the range consistent answers to aggregation queries with grouping is higher than that for the aggregation queries without grouping because for an aggregation query with grouping, AggCAvSAT needs to construct and solve twice as many WPMaxSAT instances as there are consistent groups, i.e., one for the *lub*-answer and one for the *glb*-answer per consistent group. For queries that involved the `SELECT TOP` k construct of SQL, we chose top k consistent groups ordered by one or more grouping attributes present in the `ORDER BY` clause of the query. AggCAvSAT computes the range consistent answers to each query under ten seconds except for Q_1. It took under three seconds to compute the consistent groups of Q_1, but took over forty seconds to encode the range consistent answers of the groups and over fifteen minutes to solve the corresponding WPMaxSAT instances. This is because some consistent groups have over 3M tuples and so the WPMaxSAT instances have over 600 thousand variables and over 1.3 million clauses. ConQuer took slightly over two minutes to compute the range consistent answers to Q_1. We did not include Q_1 in experiments with larger databases and higher inconsistency.

In the comparison of AggCAvSAT and ConQuer for aggregation queries with grouping on PDBench instances (Figure 6), AggCAvSAT beats ConQuer on all queries on the database instance with the lowest amount of inconsistency, but as the inconsistency grows, AggCAvSAT takes longer time to encode and solve for the consistent groups of the queries Q_3 and Q_{10}.

In Figure 7, we first plot the evaluation time of AggCAvSAT

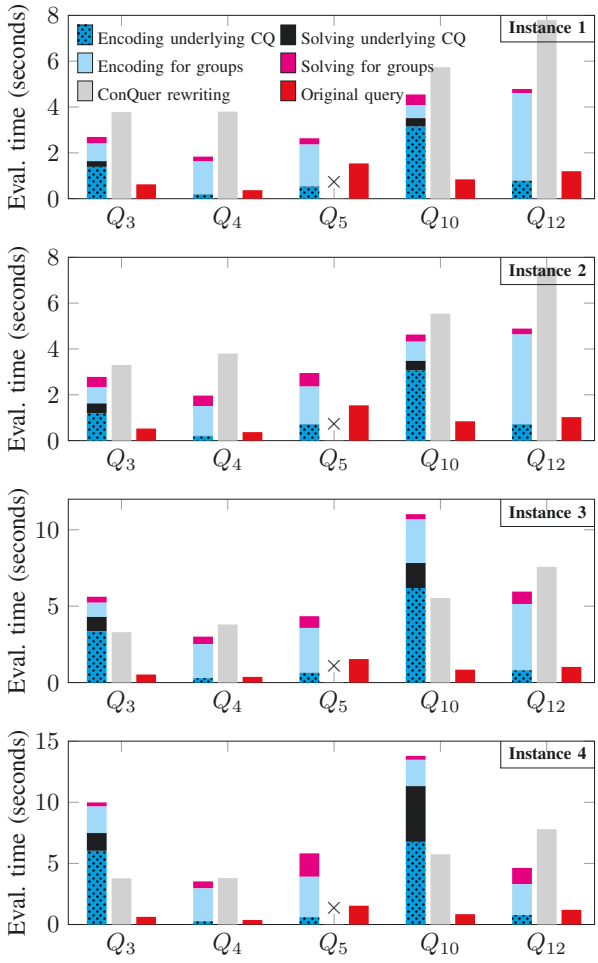

Figure 6: AggCAvSAT vs. ConQuer on PDBench instances

as the percentage of inconsistency in the data grows from 5% to 35% in the instances generated using the DBGen-based data generator. The size of the database repairs is kept constant at 1 GB (8 million tuples).

Since AggCAvSAT constructs and solves many WPMaxSAT instances having varying sizes for an aggregation query involving grouping, we also plot the overall number of SAT calls made by the solver in Figure 7. Note that the Y-axis has logarithmic scaling in the second plot of Figure 7. There are ten consistent groups in the answers to Q_3, and just five and two consistent groups in the answers to Q_5 and Q_{12} respectively. In each consistent group, the aggregation operator is applied over a much larger set of tuples in Q_5 and Q_{12} than in Q_3. As a result, the evaluation time for Q_3 is high but the number of SAT calls is comparatively less, while AggCAvSAT makes more SAT calls for Q_5 and Q_{12}, even though their consistent answers are computed much faster. The query Q_{10} requires long time to construct and solve the WPMaxSAT instances for its consistent groups due to its high selectivity and the presence of joins between four relations.

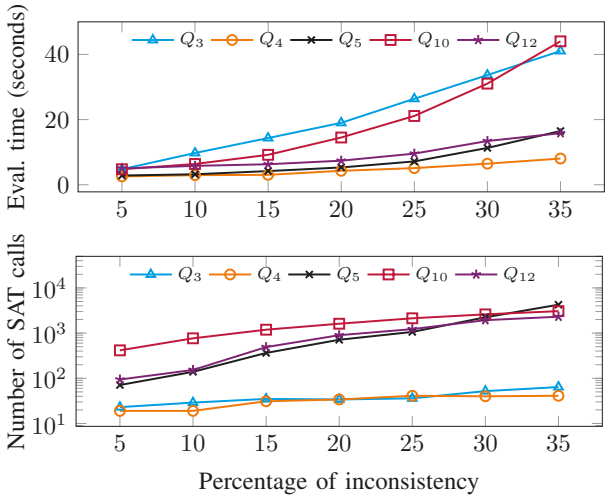

Figure 7: AggCAvSAT on TPC-H data generated using the DBGen-based tool (varying inconsistency, 1 GB repairs)

The evaluation time of computing the range consistent answers to aggregation queries with grouping increases almost linearly w.r.t. the size of the database when the percentage of inconsistency is constant (Figure 8). The second plot in Figure 8 depicts the number of SAT calls made by the solver as the size of the database grows. Due to low selectivity, the answers to Q_4 are encoded into small CNF formulas even on databases with high inconsistency or large sizes, resulting in fast evaluations.

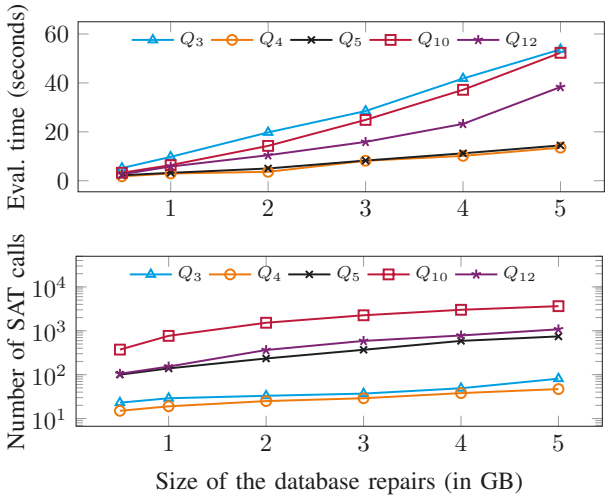

Figure 8: AggCAvSAT on TPC-H data generated using the DBGen-based tool (varying database sizes, 10% inconsistency)

5) Discussion: The experiments show that AggCAvSAT performed well across a broad range of queries and databases; it performed worse on queries with high selectivity because, in such cases, very large CNF formulas were generated. AggCAvSAT slowed down on databases with high degree of

inconsistency ($> 30\%$) and with key-equal groups of large sizes (> 15). These are rather corner cases that should not be encountered in real-world databases.

B. Experiments with Real-world Data

1) Dataset: For this set experiments, we use the schema and the data from Medigap [45], an openly available real-world database about Medicare supplement insurance in the United States. We combine the data from 2019 and 2020 to obtain a database with over 61K tuples (Table IVa). We evaluated the performance of Reduction V.1, since we consider two functional dependencies and one denial constraint on the Medigap schema, as shown in Table IVb. The actual data was inconsistent so no additional inconsistency was injected.

Table IV: Medigap real-world database

Relation	Acronym	# of attributes	# of tuples
OrgsByState	OBS	5	3872
PlansByState	PBS	18	21002
PlansByZip	PBZ	20	4748
PlanType	PT	4	2434
Premiums	PR	7	29148
SimplePlanType	SPT	4	70

(a) Medigap schema

Type	Constraint Definition	Inconsistency
FD	OBS (orgID $\rightarrow$ orgName)	2.58%
FD	PBS (addr, city, abbrev $\rightarrow$ zip)	1.5%
DC	$\forall\, t \in$ PBS (t.webAddr $\neq$ '')	0.15%

(b) Integrity constraints and inconsistency

2) Queries: We use 12 natural aggregation queries $(Q_1^m, \cdots, Q_{12}^m)$ on the Medigap database that involve the aggregation operators `COUNT(*)`, `COUNT(`A`)`, and `SUM(`A`)`. The first six queries contain no grouping, while the rest of them do. The definitions of some of these queries are in Table V, the remaining are given in [47].

Table V: Aggregation Queries on Medigap database

#	Query
Q_2^m	`SELECT COUNT(*) FROM PBZ, SPT WHERE PBZ.Description = SPT.Simple_plantype_name AND SPT.Contract_year = 2020 AND SPT.Simple_plantype = 'B'`
Q_3^m	`SELECT SUM(PBZ.Over65) FROM PBZ WHERE PBZ.State_name = 'Wisconsin' AND PBZ.County_name = 'GREEN LAKE'`
Q_4^m	`SELECT SUM(PBZ.Community) FROM PBZ WHERE PBZ.State_name = 'New York'`
Q_7^m	`SELECT SPT.Contract_year, COUNT(*) FROM SPT GROUP BY SPT.Contract_year ORDER BY SPT.Contract_year DESC`
Q_8^m	`SELECT PBZ.State_name, COUNT(*) FROM PBZ GROUP BY PBZ.State_name`
Q_{12}^m	`SELECT TOP 10 PT.Simple_plantype, COUNT(PR.Premium_range) FROM PT, PR WHERE PT.State_abbrev = PR.State_abbrev AND PT.Plan_type = PR.Plan_type AND PT.Contract_year = PR.Contract_year and PT.Contract_year = 2020 GROUP BY PT.Simple_plantype ORDER BY PT.Simple_plantype`

3) Results on Real-world Database: In Figure 9, we plot the overall time taken by AggCAvSAT to compute the range consistent answers to the twelve aggregation queries on the Medigap database. Since the Medigap schema has functional dependencies and a denial constraint, the encoding of CQA into WPMaxSAT instances is based on Reduction V.1. Consequently, the size of the CNF formulas is much larger compared to that of the ones produced by Reduction IV.1, resulting in longer encoding times. For all twelve queries, the encoding time is dominated by the time required to compute the near-violations and hence the γ-clauses. This part of the encoding time is equal for all queries, but the computation time for minimal witnesses depends on the query. The solver takes comparatively minuscule amount of time to compute the consistent answers to the underlying conjunctive query. For the queries $Q_7^m, \cdots, Q_{12}^m$, the *glb*-answer and the *lub*-answer are encoded and then solved for for each consistent group, causing high overhead. The longest evaluation time is taken by queries Q_{10}^m, Q_{12}^m, and Q_6^m since they consist of 10, 10, and 6 consistent groups, respectively.

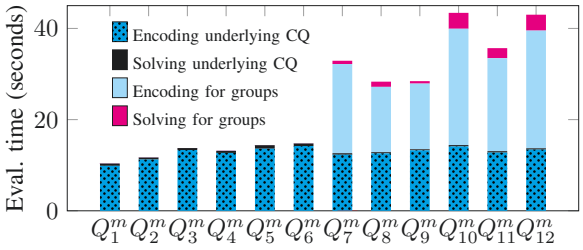

Figure 9: Evaluation on real-world aggregation queries

VII. Concluding Remarks

First, we showed that computing the range consistent answers to an aggregation query involving the `SUM(`A`)` operator can be NP-hard, even if the consistent answers to the underlying conjunctive query are SQL-rewritable. We then designed, implemented, and evaluated AggCAvSAT, a SAT-based system for computing range consistent answers to aggregation queries involving `COUNT(`A`)`, `COUNT(*)`, `SUM(`A`)`, and grouping. It is the first system able to handle aggregation queries whose range consistent answers are not SQL-rewritable. Our experimental evaluation showed that AggCAvSAT is not only competitive with systems such as ConQuer but it is also scalable. The experiments on the Medigap data showed that AggCAvSAT can handle real-world databases having integrity constraints beyond primary keys. The next step in this investigation is to first delineate the complexity of the range consistent answers to aggregation queries with the operator `AVG(`A`)` and then enhance the capabilities of AggCAvSAT to compute the range consistent answers of such aggregation queries. Finally, we note that the SAT-based methods used here are applicable to broader classes of SQL queries, such as queries with nested subqueries, as long as denial constraints are considered.

Acknowledgments: We are grateful to the ICDE 2022 reviewers for their insightful comments and pointers to the literature. Dixit was supported by a Baskin School of Engineering Dissertation-Year Fellowship and by the Center for Research in Open Source Software (CROSS) at UC Santa Cruz. Kolaitis was partially supported by NSF Award IIS: 1814152.

REFERENCES

[1] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *Proc. of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '99. New York, NY, USA: ACM, 1999, pp. 68–79. [Online]. Available: http://doi.acm.org/10.1145/303976.303983

[2] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad, "Scalar aggregation in inconsistent databases," *Theoretical Computer Science*, vol. 296, no. 3, pp. 405–434, 2003, database Theory. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397502007375

[3] M. Arenas, L. E. Bertossi, and J. Chomicki, "Answer sets for consistent query answering in inconsistent databases," *TPLP*, vol. 3, no. 4-5, pp. 393–424, 2003. [Online]. Available: https://doi.org/10.1017/S1471068403001832

[4] A. Fuxman, E. Fazli, and R. J. Miller, "ConQuer: Efficient management of inconsistent databases," in *Proc. of the 2005 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 155–166. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066176

[5] L. E. Bertossi, *Database Repairing and Consistent Query Answering*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. [Online]. Available: https://doi.org/10.2200/S00379ED1V01Y201108DTM020

[6] F. N. Afrati and P. G. Kolaitis, "Answering aggregate queries in data exchange," in *Proc. of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*. ACM, 2008, pp. 129–138. [Online]. Available: https://doi.org/10.1145/1376916.1376936

[7] E. V. Kostylev and J. L. Reutter, "Complexity of answering counting aggregate queries over dl-lite," *J. Web Semant.*, vol. 33, pp. 94–111, 2015. [Online]. Available: https://doi.org/10.1016/j.websem.2015.05.003

[8] P. Guagliardo and L. Libkin, "Making SQL queries correct on incomplete databases: A feasibility study," in *Proc. of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. ACM, 2016, pp. 211–223. [Online]. Available: https://doi.org/10.1145/2902251.2902297

[9] P. Barceló and L. E. Bertossi, "Logic programs for querying inconsistent databases," in *Practical Aspects of Declarative Languages, 5th International Symposium, PADL 2003, New Orleans, LA, USA, January 13-14, 2003, Proc.*, 2003, pp. 208–222. [Online]. Available: https://doi.org/10.1007/3-540-36388-2_15

[10] J. Chomicki, J. Marcinkowski, and S. Staworko, "Hippo: A system for computing consistent answers to a class of sql queries," in *Advances in Database Technology - EDBT 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 841–844.

[11] A. Fuxman, D. Fuxman, and R. J. Miller, "ConQuer: A system for efficient querying over inconsistent databases," in *Proc. of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05. VLDB Endowment, 2005, pp. 1354–1357. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083592.1083774

[12] G. Greco, S. Greco, and E. Zumpano, "A logical framework for querying and repairing inconsistent databases," *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, no. 6, pp. 1389–1408, Nov. 2003. [Online]. Available: https://doi.org/10.1109/TKDE.2003.1245280

[13] P. G. Kolaitis, E. Pema, and W. Tan, "Efficient querying of inconsistent databases with binary integer programming," *PVLDB*, vol. 6, no. 6, pp. 397–408, 2013. [Online]. Available: http://www.vldb.org/pvldb/vol6/p397-tan.pdf

[14] M. Manna, F. Ricca, and G. Terracina, "Consistent query answering via ASP from different perspectives: Theory and practice," *CoRR*, vol. abs/1107.4570, 2011. [Online]. Available: http://arxiv.org/abs/1107.4570

[15] M. C. Marileo and L. E. Bertossi, "The consistency extractor system: Answer set programs for consistent query answering in databases," *Data Knowl. Eng.*, vol. 69, no. 6, pp. 545–572, 2010. [Online]. Available: https://doi.org/10.1016/j.datak.2010.01.005

[16] A. A. Dixit and P. G. Kolaitis, "A SAT-based system for consistent query answering," in *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proc.*, ser. Lecture Notes in Computer Science, vol. 11628. Springer, 2019, pp. 117–135. [Online]. Available: https://doi.org/10.1007/978-3-030-24258-9_8

[17] J. Chomicki, J. Marcinkowski, and S. Staworko, "Computing consistent query answers using conflict hypergraphs," in *Proc. of the Thirteenth ACM International Conference on Information and Knowledge Management*, ser. CIKM '04. New York, NY, USA: ACM, 2004, pp. 417–426. [Online]. Available: http://doi.acm.org/10.1145/1031171.1031254

[18] J. Davies and F. Bacchus, "Solving MAXSAT by solving a sequence of simpler SAT instances," in *Principles and Practice of Constraint Programming – CP 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 225–239.

[19] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *2008 IEEE 24th International Conference on Data Engineering*, 2008, pp. 983–992.

[20] A. A. Dixit and P. G. Kolaitis, "Consistent answers of aggregation queries using SAT solvers," *CoRR*, vol. abs/2103.03314, 2021. [Online]. Available: https://arxiv.org/abs/2103.03314

[21] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "Holoclean: Holistic data repairs with probabilistic inference," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1190–1201, Aug. 2017. [Online]. Available: https://doi.org/10.14778/3137628.3137631

[22] S. Giannakopoulou, M. Karpathiotakis, and A. Ailamaki, "Query-driven repair of functional dependency violations," in *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 2020, pp. 1886–1889. [Online]. Available: https://doi.org/10.1109/ICDE48307.2020.00195

[23] ——, "Cleaning denial constraint violations through relaxation," in *Proc. of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 805–815. [Online]. Available: https://doi.org/10.1145/3318464.3389775

[24] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995. [Online]. Available: http://webdam.inria.fr/Alice/

[25] A. Fuxman and R. J. Miller, "First-order query rewriting for inconsistent databases," *J. Comput. Syst. Sci.*, vol. 73, no. 4, pp. 610–635, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jcss.2006.10.013

[26] J. Wijsen, "Consistent query answering under primary keys: A characterization of tractable queries," in *Proc. of the 12th International Conference on Database Theory*, ser. ICDT '09. New York, NY, USA: ACM, 2009, pp. 42–52. [Online]. Available: http://doi.acm.org/10.1145/1514894.1514900

[27] ——, "On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases," in *Proc. of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '10. New York, NY, USA: ACM, 2010, pp. 179–190. [Online]. Available: http://doi.acm.org/10.1145/1807085.1807111

[28] ——, "A remark on the complexity of consistent conjunctive query answering under primary key violations," *Information Processing Letters*, vol. 110, no. 21, pp. 950 – 955, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020019010002395

[29] P. G. Kolaitis and E. Pema, "A dichotomy in the complexity of consistent query answering for queries with two atoms," *Inf. Process. Lett.*, vol. 112, no. 3, pp. 77–85, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.ipl.2011.10.018

[30] P. Koutris and J. Wijsen, "Consistent query answering for primary keys," *SIGMOD Rec.*, vol. 45, no. 1, pp. 15–22, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/2949741.2949746

[31] ——, "Consistent query answering for self-join-free conjunctive queries under primary key constraints," *ACM Trans. Database Syst.*, vol. 42, no. 2, pp. 9:1–9:45, Jun. 2017. [Online]. Available: http://doi.acm.org/10.1145/3068334

[32] M. Y. Vardi, "Symbolic techniques in propositional satisfiability solving," in *Theory and Applications of Satisfiability Testing - SAT 2009*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 2–3.

[33] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proc. of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, p. 399–404.

[34] "CaDiCaL simplified satisfiability solver," http://fmv.jku.at/cadical/.

[35] M. J. H. Heule, O. Kullmann, and V. W. Marek, "Solving and verifying the boolean pythagorean triples problem via cube-and-conquer," in *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th Intern. Conference, Bordeaux, France, July 5-8, 2016, Proc.*, ser. Lecture

Notes in Computer Science, vol. 9710. Springer, 2016, pp. 228–245. [Online]. Available: https://doi.org/10.1007/978-3-319-40970-2_15

[36] P. Oostema, R. Martins, and M. Heule, "Coloring unit-distance strips using SAT," in *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, ser. EPiC Series in Computing, vol. 73. EasyChair, 2020, pp. 373–389. [Online]. Available: https://easychair.org/publications/paper/69T4

[37] A. Fuxman, "Efficient query processing over inconsistent databases," Ph.D. dissertation, Department of Computer Science, University of Toronto, 2007.

[38] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972, pp. 85–103. [Online]. Available: https://doi.org/10.1007/978-1-4684-2001-2_9

[39] C.-M. Li, Z. Zhu, F. Manyà, and L. Simon, "Minimum satisfiability and its applications," in *Proc. of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, ser. IJCAI'11. AAAI Press, 2011, p. 605–610.

[40] A. Kügel, "Natural max-sat encoding of min-sat," in *Revised Selected Papers of the 6th International Conference on Learning and Intelligent Optimization - Volume 7219*, ser. LION 6. Berlin, Heidelberg: Springer-Verlag, 2012, p. 431–436.

[41] Z. Zhu, C.-M. Li, F. Manyà, and J. Argelich, "A new encoding from minsat into maxsat," in *Principles and Practice of Constraint Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 455–463.

[42] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, L. Libkin, Ed. ACM, 2007, pp. 31–40. [Online]. Available: https://doi.org/10.1145/1265530.1265535

[43] G. Karvounarakis and T. J. Green, "Semiring-annotated data: queries and provenance?" *SIGMOD Rec.*, vol. 41, no. 3, pp. 5–14, 2012. [Online]. Available: https://doi.org/10.1145/2380776.2380778

[44] Y. Amsterdamer, D. Deutch, and V. Tannen, "Provenance for aggregate queries," in *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, M. Lenzerini and T. Schwentick, Eds. ACM, 2011, pp. 153–164. [Online]. Available: https://doi.org/10.1145/1989284.1989302

[45] "Medigap database for medicare health and drug plans," https://www.medicare.gov/download/downloaddb.asp.

[46] A. Dixit and P. G. Kolaitis, "CAvSAT: Answering aggregation queries over inconsistent databases via SAT solving," in *Proc. of the 2021 ACM International Conference on Management of Data (SIGMOD) - Demonstration Track*, 2021, to appear.

[47] A. A. Dixit and P. G. Kolaitis, "Consistent answers of aggregation queries using SAT solvers," *CoRR*, vol. abs/2103.03314, 2021. [Online]. Available: http://arxiv.org/abs/2103.03314