# SIGMOD: G: CAvSAT: A System for Query Answering over Inconsistent Databases

Akhil A. Dixit (Advisor: Prof. Phokion G. Kolaitis)
akadixit@ucsc.edu
University of California, Santa Cruz

**ACM Reference Format:**
Akhil A. Dixit (Advisor: Prof. Phokion G. Kolaitis). 2020. SIGMOD: G: CAvSAT: A System for Query Answering over Inconsistent Databases. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 PROBLEM AND MOTIVATION

Managing inconsistencies in databases is an old, but recurring, problem. An *inconsistent* database is a database that violates one or more integrity constraints, such as key constraints or functional dependencies. In the real-world, inconsistent databases arise due to various reasons and in several different contexts, including data warehousing and information integration, where dealing with inconsistency is regarded as a key challenge [10]. Inconsistencies are more of a norm than an exception, and they play a major role in a more general and broadly used term *poor quality data*, which, according to a report from IBM, cost $3.1 trillion to the US economy in 2016 [2]. *Data cleaning* is the main approach towards managing inconsistent databases (see the survey [26]). Some data cleaning approaches derive a consistent database by resolving all conflicts that exist in a given inconsistent database. This process often relies on clustering techniques, heuristics, and/or on particular domain knowledge. While data cleaning makes it possible to derive a single consistent database, this approach often involves making arbitrary choices for removing inconsistencies, and thus it is often *ad hoc*; for example, if a book has two different ISBN numbers stored in a database, there may not be an apriori reason to decide which one of the two should be removed.

An alternative, and arguably more principled, approach is the framework of *database repairs* and *consistent query answering (CQA)*, introduced in [5]. Here, the inconsistencies are handled at query time by considering all possible repairs of the inconsistent database, where a *repair* of an inconsistent database $I$ is a consistent database $r$ that differs from $I$ in a "minimal" way. The main algorithmic problem in this framework is to compute the *consistent answers* $\textsc{Cons}(q, I)$ *of a query* $q$ *on a given database* $I$, that is, the tuples that lie in the intersection of the results of $q$ applied on each repair of $I$. Computing the consistent answers of a query $q$ on $I$ can be computationally harder than simply evaluating $q$ on $I$, because an inconsistent database may have exponentially many repairs. In fact, the consistent answers exhibit a variety of computational behaviors even for simple conjunctive queries under key constraints, from SQL-rewritability to coNP-completeness.

There have been several attempts of building a system for CQA, including [7, 9, 15, 22, 23, 25, 28, 32, 33], but most of them remained as academic prototypes for various reasons. In particular, the ConQuer system [22, 23] is tailored to queries in the subclass (called $C_{forest}$) of SQL-rewritable queries. Other systems use logic programming [9, 25], compact representations of repairs [14], or reductions to solvers. Specifically, the system in [32] uses reductions to answer set programming, while the EQUIP system in [28] uses reductions to binary integer programming. It is fair to say, however, no comprehensive and scalable system for consistent query answering exists at present; this state of affairs has impeded the broader adoption of the framework of repairs and consistent answers as a principled alternative to data cleaning. The primary challenge in building a practical CQA system appears to be the high computational complexity of the consistent answers, and therefore the scalability of the system. In this work, we propose a novel approach based on *boolean Satisfiability* (SAT) to build a CQA system that scales well with the data, and supports broad classes of practical queries and integrity constraints. We also report experimental results obtained using our prototype CQA system CAvSAT (Consistent Answering via Satisfiability), implemented using this approach.

## 2 BACKGROUND AND RELATED WORK

**Integrity Constraints and Database Queries.** Relational database schemas are often accompanied by a set of integrity constraints that impose semantic restrictions on the allowable instances. A *functional dependency* $\vec{x} \rightarrow \vec{y}$ on a relation $R$ is an integrity constraint asserting that if two tuples agree on the attributes in $\vec{x}$, then they must also agree on the attributes in $\vec{y}$. Typically, the set of all attributes of relation $R$ is denoted by $Attr(R)$. A *primary key* (or simply, *key*) is a minimal subset $\vec{x}$ of $Attr(R)$ such that the functional dependency $\vec{x} \rightarrow Attr(R)$ holds. *Denial constraints* is a broad class of integrity constraints that contains keys and functional dependencies as special cases. A denial constraint prohibits a set of tuples that satisfy certain conditions from appearing together in a database instance; for example, a restriction on a university database that each instructor can teach at most three courses per semester can be expressed as a denial constraint but not using keys or functional dependencies alone.

*Conference'17, July 2017, Washington, DC, USA*
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

As is well known, first-order logic has been successfully used as a database query language [16, 17]. In fact, it forms the core of SQL, the main commercial database query language. A *conjunctive query* is a query expressible by a first-order formula of the form $q(\vec{z}) := \exists \vec{w}\ (R_1(\vec{x_1}) \wedge \ldots \wedge R_m(\vec{x_m}))$, where each $\vec{x_i}$ is a tuple consisting of variables and constants, $\vec{z}$ and $\vec{w}$ are tuples of variables, and the variables in $\vec{x_1}, \ldots, \vec{x_m}$ appear in exactly one of $\vec{z}$ and $\vec{w}$. A conjunctive query with $k$ free variables in $\vec{z}$ is a $k$-ary query, while a conjunctive query with no free variables (i.e., all variables are existentially quantified) is a boolean query. Conjunctive queries are also known as *select-project-join* (SPJ) queries and are among the most frequently asked queries in databases. For example, the binary conjunctive query $q(s, t) := \exists c\ (\text{Enrolls}(s, c) \wedge \text{Teaches}(t, c))$ returns the set of all pairs $(s, t)$ such that student $s$ is enrolled in a course taught by teacher $t$, while the boolean conjunctive query $q() := \exists x, y, z\ (E(x, y) \wedge E(y, z) \wedge E(z, x))$ tests whether a graph with an edge relation $E$ contains a triangle. In the real-world, database queries often involve aggregation functions such as `MIN`, `MAX`, `SUM`, `COUNT`, or `AVG` on top of a conjunctive query. For example, an aggregation query $Q :=$ `SELECT DISTINCT COUNT(`$s$`) FROM` $(\text{Enrolls}(s, c) \wedge \text{Teaches}(\text{'Lee'}, c))$ returns the number of students enrolled in the courses taught by teacher Lee.

**Database Repairs and Consistent Answers.** Let $\mathcal{R}$ be a database schema and $\Sigma$ be a set of integrity constraints on $\mathcal{R}$. An $\mathcal{R}$-instance $I$ is *consistent* if $I \models \Sigma$, i.e., $I$ satisfies every constraint in $\Sigma$; otherwise, $I$ is *inconsistent*. A *repair* of an inconsistent instance $I$ w.r.t. $\Sigma$ is a consistent instance $J$ that differs from $I$ in a "minimal" way. Different notions of minimality give rise to different types of repairs (see [11] for a comprehensive survey). Here, we focus on *subset repairs*, the most extensively studied type of repairs. An instance $J$ is a *subset repair* of an instance $I$ if $J \subseteq I$ (where $I$ and $J$ are viewed as sets of facts), $J \models \Sigma$, and there exists no instance $J'$ such that $J' \models \Sigma$ and $J \subset J' \subset I$. Arenas et al. [5] used repairs to give rigorous semantics to query answering on inconsistent databases. Specifically, assume that $q$ is a query, $I$ is an $\mathcal{R}$-instance, and $\vec{t}$ is a tuple of values. We say that $\vec{t}$ is a *consistent answer* to $q$ on $I$ w.r.t. $\Sigma$ if $\vec{t} \in q(J)$, for every repair $J$ of $I$. We write $\textsc{Cons}(q, I, \Sigma)$ to denote the set of all *consistent answers* to $q$ on $I$ w.r.t. $\Sigma$, i.e.,

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

If $\Sigma$ is a fixed set of integrity constraints and $q$ is a fixed first-order query, then the main computational problem associated with the consistent answers is: given an instance $I$, compute $\textsc{Cons}(q, I, \Sigma)$. If $q$ is a boolean conjunctive query, then computing the consistent answers becomes the decision problem $\textsc{certainty}(q, \Sigma)$: given an instance $I$, is $q$ true on every repair $J$ of $I$ w.r.t. $\Sigma$?

**Computational Complexity of Consistent Answers.** By now, there has been an extensive study of CQA of conjunctive queries [11, 24, 27, 36, 37]. If $\Sigma$ is a fixed finite set of denial constraints and $q$ is a $k$-ary conjunctive query, where $k \geq 1$, then computing $\textsc{Cons}(q, I, \Sigma)$ given an instance $I$ is in coNP. Even for key constraints and boolean conjunctive queries, $\textsc{certainty}(q, \Sigma)$ exhibits a variety of behaviors within coNP. The most definitive result to date is a *trichotomy* theorem [29–31]. This trichotomy theorem asserts that if $q$ is a self-join-free (no repeated relation symbols) 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. Moreover, there is a quadratic algorithm to decide, given such a query, which of the three cases of the trichotomy holds. It remains an open problem whether or not this trichotomy extends to arbitrary boolean conjunctive queries and to arbitrary denial constraints.

**Boolean Satisfiability and SAT Solvers.** *Boolean Satisfiability* (SAT) is the prototypical and the most widely studied NP-complete problem [18]. SAT asks the following: *given a boolean formula, is it satisfiable?* There has been extensive research about SAT-solving (see [12] for a survey), and this field of study has witnessed tremendous progress in the last few years (often referred as "SAT Revolution" [35]). Today's SAT solvers are capable of solving SAT instances with millions of clauses and variables within a very short period of time. This success has resulted into widespread use of SAT solvers in industry as general-purpose problem-solving tools. Many real-world problems from a variety of domains such as scheduling, protocol design, software verification, model checking, and more can be naturally encoded as SAT instances, and solved quickly using solvers such as Glucose [8] and CaDiCaL [1] among others. Typically, a SAT solver takes a boolean formula in *Conjunctive Normal Form (CNF)* as an input and outputs a satisfying assignment (if one exists) or tells the formula is unsatisfiable. In an optimization variant of SAT, the clauses of the formula are assigned integer weights and the goal is to find an assignment that maximizes the sum of the weights of the satisfied clauses. This variant is known as Weighted MaxSAT, and modern solvers such as MaxHS [19] can solve Weighted MaxSAT instances very efficiently in practice.

## 3 APPROACH AND UNIQUENESS

We have been developing a comprehensive and scalable SAT-based system for CQA, namely, CAvSAT (Consistent Answers via Satisfiability). Figure 1 depicts its modular architecture. In the query pre-processor module, we leverage the extensive literature about the computational complexity of the consistent answers (e.g., [13, 27, 31, 38, 39]) to "route" the query to different modules tailored to handle the consistent answers of queries in a particular complexity class (SQL-rewritable, PTIME computable but not SQL-rewritable, and coNP-complete). The distinctive feature of CAvSAT is the SAT-solving module. Ours is the first CQA system that takes advantage of the progress made by the SAT-solving technology over the past few years and applies it to CQA. CAvSAT is also the first system to handle aggregation queries whose consistent answers (w.r.t. range semantics) are not SQL-rewritable. Furthermore, CAvSAT is flexible and extensible; when new classes of queries whose consistent answers are SQL-rewritable or tractable are identified, the system will be extended to incorporate these findings in the pre-processing stage, thus using a potentially more efficient evaluation strategy for computing the consistent answers of the queries at hand. Also, as the SAT-solving technology progresses, newer solvers could be used to leverage such advances. The CAvSAT source code is available at the GitHub repository *https://github.com/uccross/cavsat* via a BSD-style open-source license.

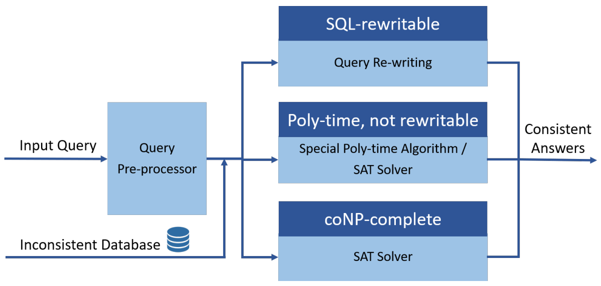

**Figure 1: Modular architecture of CAvSAT**

In what follows, we describe a reduction from CQA to Weighted MaxSAT, which is used by CAvSAT in its SAT-solving module to compute the consistent answers to unions of conjunctive queries (UCQs) over databases that are inconsistent w.r.t. a set of arbitrary denial constraints.

## 3.1 Consistent Answers to UCQs via SAT Solving

Let $\mathcal{R}$ be a database schema, $\Sigma$ be a fixed finite set of denial constraints over $\mathcal{R}$, and $Q$ be a fixed union of conjunctive queries. Note that computing consistent answers to unions of conjunctive queries under denial constraints is still in coNP, but the consistent answers to a union $Q := q_1 \cup \ldots \cup q_k$ of conjunctive queries $q_1, \ldots, q_k$ is not, in general, equal to the union of the consistent answers to $q_1, \ldots, q_k$. In what follows, we give a polynomial-time reduction from Cons($Q$, $I$, $\Sigma$) to UnSAT. The reduction we give here relies on the notions of *minimal violations* and *near-violations* to the set of denial constraints that we introduce next.

Definition 1. ***Minimal violation.*** *Assume that $\Sigma$ is a set of denial constraints, $I$ is an $\mathcal{R}$-instance, and $S$ is a sub-instance of $I$. We say that $S$ is a minimal violation to $\Sigma$, if $S \not\models \Sigma$ and for every set $S' \subset S$, we have that $S' \models \Sigma$.*

Definition 2. ***Near-violation.*** *Assume that $\Sigma$ is a set of denial constraints, $I$ is an $\mathcal{R}$-instance, $S$ is a sub-instance of $I$, and $f$ is a fact of $I$. We say that $S$ is a near-violation to $\Sigma$ w.r.t. $f$, if $S \models \Sigma$ and $S \cup \{f\}$ is a minimal violation to $\Sigma$. Special case: if $\{f\}$ itself is a minimal violation to $\Sigma$, then 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.*

Reduction 1. *Given an $\mathcal{R}$-instance $I$, compute the following sets:*

- *$\mathcal{V}$: the set of minimal violations to $\Sigma$ on $I$.*
- *$\mathcal{N}^i$: the set of near-violations to $\Sigma$, on $I$, w.r.t. each tuple $f_i \in I$.*
- *$\mathcal{A}$: the set of answers to $Q$ on $I$, i.e., the set $Q(I)$.*
- *$\mathcal{W}^{\mathbf{a}}$: the set of all minimal witnesses to $Q[\mathbf{a}]$ on $I$, for each $\mathbf{a} \in \mathcal{A}$.*

*For each fact $f_i \in I$, introduce a boolean variable $x_i$. For the auxiliary fact $f_{true}$, let $x_{true} = true$. For each $N^i_j \in \mathcal{N}^i$, introduce a boolean variable $y^i_j$, and for each $\mathbf{a} \in \mathcal{A}$, introduce a boolean variable $p_{\mathbf{a}}$.*

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 $\mathbf{a} \in \mathcal{A}$ and for each $W^{\mathbf{a}}_j \in \mathcal{W}^{\mathbf{a}}$, construct a clause $\beta^{\mathbf{a}}_j = \left(\bigvee_{f_i \in W^{\mathbf{a}}_j} \neg x_i\right) \vee \neg p_{\mathbf{a}}$.*
3. *For each $f_i \in I$, construct a clause $\gamma_i = x_i \vee \left(\bigvee_{N^i_j \in \mathcal{N}^i} y^i_j\right)$.*
4. *For each variable $y^i_j$, construct an expression $\theta^i_j = y^i_j \leftrightarrow \left(\bigwedge_{f_d \in N^i_j} x_d\right)$.*
5. *Construct a boolean formula $\phi$ as follows:*
$$\phi = \bigwedge_{i=1}^{|\mathcal{V}|} \alpha_i \wedge \left(\bigwedge_{\mathbf{a} \in \mathcal{A}}\left(\bigwedge_{j=1}^{|\mathcal{W}^{\mathbf{a}}|} \beta^{\mathbf{a}}_j\right)\right) \wedge \left(\bigwedge_{i=1}^{|I|}\left(\left(\bigwedge_{j=1}^{|\mathcal{N}^i|} \theta^i_j\right) \wedge \gamma_i\right)\right)$$

Proposition 1. *Let $\phi$ be the boolean formula constructed using Reduction 1. There exists a satisfying assignment to $\phi$ in which a variable $p_{\mathbf{a}}$ is set to true if and only if $\mathbf{a} \notin$ Cons$(Q, I, \Sigma)$.*

It follows from Proposition 1 that, if we find all $p_{\mathbf{a}}$-variables such that they get set to true in at least one satisfying assignment to $\phi$, we can safely eliminate all their corresponding $\mathbf{a}$-answers for being inconsistent. In practice, this can be done fast via a simple iterative algorithm mentioned in [20], that relies on a deployment of a state-of-the-art Partial MaxSAT solver or a Weighted MaxSAT solver to find satisfying assignments to $\phi$.

**Optimizations in Reduction 1.** The preceding reduction is generic, in the sense that it supports a broad class of integrity constraints, namely, arbitrary denial constraints. If we restrict the database schema to have only key constraints, however, a more efficient reduction can be applied [20]. Moreover, in real-life applications, a large portion of the inconsistent database is often consistent, and we leverage this fact by efficiently pre-computing the consistent answers that come from the consistent part of the database. This can be done with simple SQL queries that involve grouping on the key attributes of each relation. By doing so, we need boolean variables corresponding to only those tuples that contribute to the answers that are not pre-computed. This significantly reduces the size of the SAT instances produced by Reduction 1.

## 3.2 Consistent Answers to Aggregation Queries via SAT Solving

Let $\mathcal{R}$ be a database schema with one key constraint per relation, and $Q$ be the aggregation query

$$Q := \texttt{SELECT } f \texttt{ FROM } T(\vec{u}, w),$$

where $f$ is one of COUNT($*$), COUNT($w$), SUM($w$), MIN($w$), or MAX($w$), and $T(\vec{u}, w)$ is a relation expressed by a self-join-free conjunctive query on $\mathcal{R}$. Unlike a conjunctive query, an aggregation query is likely to return different answers in different repairs of an inconsistent database, and there may not be a single consistent answer to it as per the preceding definition of $\textsc{Cons}(q, I, \Sigma)$. In order to obtain more informative answers in such a case, Arenas et al. [6] introduced an extended notion of consistent query answers, called *range semantics*. The consistent answer to an aggregation query $Q$ on an inconsistent database $I$ w.r.t. the range semantics is a pair $\{glb, lub\}$ of answers, i.e., the tightest range of answers, such that, $glb \leq q(J) \leq lub$ holds for every repair $J$ of $I$.

Ongoing work involves enabling CAvSAT to compute consistent answers to aggregation queries using SAT-solving. In what follows, we show how to compute the consistent answers (w.r.t. range semantics) to aggregation queries involving `MIN` function over databases with one key constraint per relation. Slight modifications can be made to Reduction 2 for the queries with other aggregation functions `MAX`, `SUM`, and `COUNT`.

Let $Q :=$ `SELECT MIN`$(w)$ `FROM` $T(\vec{u}, w)$. Simply evaluating the query $Q$ on $I$ returns the $glb$-answer in $\textsc{Cons}(Q, I, \Sigma)$. To see why this is true, observe that, *i*) no repair of $I$ can produce a smaller answer to $Q$ than the one returned by $Q(I)$, and *ii*) there exists a repair $R$ of $I$ such that $Q(R) = Q(I)$. For computing the *lub*-answer in $\textsc{Cons}(Q, I, \Sigma)$, we give a polynomial-time reduction from $\textsc{Cons}(Q, I, \Sigma)$ to Weighted Partial MaxSAT.

Reduction 2. *Given an $\mathcal{R}$-instance $I$, construct a* Weighted Partial MaxSAT *instance $\phi$ as follows. For each tuple $f_i$ of $I$, introduce a boolean variable $x_i$. Let $\mathcal{G}$ be the set of key-equal groups of tuples of $I$, and $\mathcal{W} = \{W_1, \cdots, W_m\}$ denote the set of minimal witnesses to a conjunctive query $q$ on $I$, where $q(w) := \exists \vec{u}\ T(\vec{u}, w) \wedge (w$ `IS NOT NULL`$)$. For the set $\mathcal{W}$, assume for $1 \leq i \leq m$, we have that $q(W_i) \geq q(W_{i+1})$.*

- *For each $G_j \in \mathcal{G}$, construct a hard clause $\alpha_j = \bigvee_{f_i \in G_j} x_i$.*
- *Let $c = 1$, $sum = 0$. For $j = 1$ to $m$, do the following.*
  - *Construct a clause $\beta_j = \bigvee_{f_i \in W_j} \neg x_i$, and let its weight be $c$.*
  - *$sum = sum + c$.*
  - *If ($j > 1$ and $q(W_j) < q(W_{j-1})$) then $c = sum + 1$.*
- *Construct a* Weighted Partial MaxSAT *instance*
$$\phi = \left(\bigwedge_{j=1}^{|\mathcal{G}|} \alpha_j\right) \wedge \left(\bigwedge_{j=1}^{|\mathcal{W}|} \beta_j\right).$$

Proposition 2. *Let $\hat{\phi}$ be a maximum satisfying assignment to the* Weighted Partial MaxSAT *instance $\phi$ constructed using Reduction 2. Let $\beta_j$ be a clause such that $\hat{\phi}(\beta_j) = false$, and there is no $j' < j$ such that $\hat{\phi}(\beta_{j'}) = false$. Then, $q(W_j)$ is the lub-answer in $\textsc{Cons}(Q)$ on $I$.*

## 4 RESULTS AND CONTRIBUTIONS

In the first set of experiments, we compared the performance of the SAT-solving module of CAvSAT against the SQL-rewritings of seven conjunctive queries chosen from the literature [28], over a synthetically generated database (1 million tuples/relation with 10% inconsistency w.r.t. primary keys). For queries $q_1, \ldots, q_7$, we computed the SQL-rewritings using the algorithm of Koutris and Wijsen [31] (referred as KW-SQL-rewriting). Since the queries $q_1, \ldots, q_4$ happen to be in the class $C_{forest}$, we computed additional SQL-rewritings for them using the algorithm implemented in the CQA system ConQuer [22] (referred as ConQuer-SQL-rewriting). Figure 2 shows that for the queries $q_1, \ldots, q_4$ in the class $C_{forest}$, the performance of the SAT-solving module was slightly worse, but comparable, to their ConQuer-SQL-rewritings. For all seven queries $q_1, \ldots, q_7$, however, SAT-based approach significantly outperformed KW-SQL-rewritings, as the database engine hit the two hours timeout while evaluating each KW-SQL-rewriting.

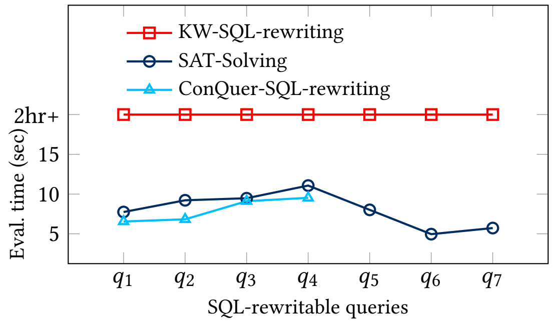

**Figure 2: Performance of the SAT-solving module of CAvSAT against the SQL-rewriting approaches**

Next, we considered fourteen additional conjunctive queries whose consistent answers are coNP-complete or in P but not SQL-rewritable. Figure 3 shows that the time required for constructing the Weighted MaxSAT instances dominates over the time taken to solve them and eliminate inconsistent answers. The solver took comparatively more time for the queries that have higher number of free variables or joins. Observe that even for the queries with intractable consistent answers, the performance of the SAT-solving module did not degrade too much compared to the SQL-rewritable queries.

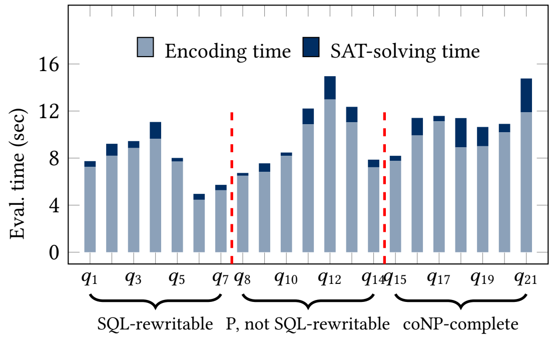

**Figure 3: Performance of the SAT-solving module for conjunctive queries of varying data complexity**

In the next set of experiments, we evaluated the performance of the SAT-solving module on real-world data having key constraints on each relation, along with one functional dependency (that is not a key constraint). The data used are about inspections of food establishments in New York and Chicago, and are taken from [4] and [3], and its size was up to 343k tuples/relation with up to 25% inconsistency. Part of this data have been previously used for evaluating data cleaning systems, such as HoloClean [34]. We evaluated the performance of the SAT-solving module on five conjunctive queries $Q_1, \cdots, Q_5$ and one union $Q_6$ of conjunctive queries. The optimizations mentioned in Section 3.1 were not applied during this set of experiments. We observed that the SAT solver took more time to compute consistent answers to query $Q_3$ due to high number of joins, and the union $Q_6$ due to high number of minimal witnesses (Figure 4). All aforementioned experiments were carried out on a machine running on Intel Core i7 2.7 GHz, 64 bit Ubuntu 16.04, with 8 GB RAM. The precise definitions of the queries used in the experiments and additional experimental results can be found in our paper [21].

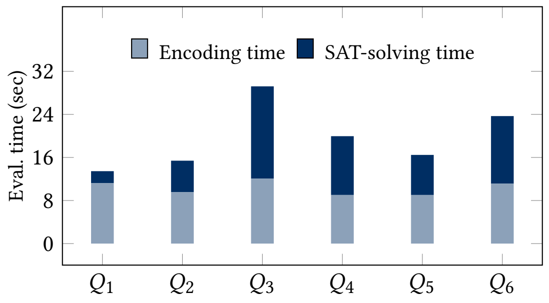

**Figure 4: Performance of the SAT-solving module on real-world data**

The findings from Figure 2 suggest a potential difference between theory and practice, since the study of SQL-rewritability of the consistent answers was motivated from having an efficient evaluation of consistent answers using the database engine alone. To investigate further, we conducted experiments on large TPC-H benchmark databases (12 million tuples/relation with 5% inconsistency w.r.t. primary keys) and fifteen TPC-H-inspired queries, on Microsoft Azure SQL Database (200 DTUs). Queries $Q_{t1}, \cdots, Q_{t9}$ were in $C_{forest}$, thus we computed their consistent answers using ConQuer's algorithm [22]. Table 1 shows the comparison of the evaluation time in seconds for computing the consistent answers. The SAT-solving approach outperformed the ConQuer SQL-rewriting for five out of nine queries, once again highlighting the gap between theory and practice. For queries $Q_{t3}$ and $Q_{t8}$, however, the reason behind the poor performance of the SAT-solving module appears to be the high number of joins between relatively large relations causing significant increase in the size of the SAT instances. The column potential answers refers to the time taken to directly evaluate the input query on the inconsistent database and helps understand the overhead incurred due to computing consistent answers using both SAT-based and SQL-rewriting approaches.

**Conclusions.** We designed and implemented CAvSAT, the first SAT-based system for consistent query answering. CAvSAT leverages a set of natural reductions from the complement of CQA to SAT and its variants. The system is currently capable of handling unions of conjunctive queries and arbitrary denial constraints. Ongoing work involves computing consistent answers to aggregation queries w.r.t. range semantics. Our experimental evaluation provides evidence that a SAT-based approach is indeed promising, and can give rise to a comprehensive and scalable system for CQA. The next step in this investigation is to carry out an extensive comparative evaluation of CAvSAT with other systems for CQA that can handle the queries whose consistent answers are not SQL-rewritable, in particular, with systems that use reduction-based methods [28, 32]. A longer-term and more ambitious endeavor is to carry out a comparison between CAvSAT and a data cleaning system such as HoloClean [34] on various fronts including performance and semantic guarantees. This is indeed a challenging endeavor because it will require first to develop a methodology for carrying out a meaningful comparison between two fundamentally different approaches to handle inconsistent databases, namely, data cleaning and consistent query answering.

**Table 1: Performance of the SAT-solving module over TPC-H data (for $C_{forest}$ queries)**

| Query | Partial MaxSAT | ConQuer Rewriting | Potential Answers |
|---|---|---|---|
| $Q_{t1}$ | **3.0s** | 59.5s | 1.8s |
| $Q_{t2}$ | **0.8s** | 1.1s | 0.1s |
| $Q_{t3}$ | 13.5s | **4.4s** | 0.5s |
| $Q_{t4}$ | **0.4s** | 1.8s | 0.2s |
| $Q_{t5}$ | **7.2s** | 8.2s | 1.7s |
| $Q_{t6}$ | **7.7s** | 8.1s | 0.9s |
| $Q_{t7}$ | 2.0s | **1.2s** | 0.1s |
| $Q_{t8}$ | 24.4s | **11.3s** | 1.8s |
| $Q_{t9}$ | 3.4s | **1.9s** | 2.0s |

**Table 2: Performance of the SAT-solving module over TPC-H data (for non-$C_{forest}$ queries)**

| Query | Partial MaxSAT | ConQuer Rewriting | Potential Answers |
|---|---|---|---|
| $Q_{t10}$ | 1.5s | - | 1.1s |
| $Q_{t11}$ | 1.1s | - | 0.9s |
| $Q_{t12}$ | 2.1s | - | 0.9s |
| $Q_{t13}$ | 1.6s | - | 0.8s |
| $Q_{t14}$ | 0.4s | - | 0.5s |
| $Q_{t15}$ | 6.0s | - | 1.0s |

## REFERENCES

[1] CaDiCaL simplified satisfiability solver. http://fmv.jku.at/cadical/.
[2] The four V's of big data. https://www.ibmbigdatahub.com/infographic/four-vs-big-data.
[3] Food inspections, city of Chicago. https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5, Aug 2011.
[4] New york city restaurant inspection results, department of health and mental hygiene (DOHMH). https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j, Aug 2014.
[5] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '99, pages 68–79, New York, NY, USA, 1999. ACM.

[6] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theoretical Computer Science*, 296(3):405–434, 2003. Database Theory.

[7] M. Arenas, L. E. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

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

[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, Proceedings*, pages 208–222, 2003.

[10] P. A. Bernstein and L. M. Haas. Information integration in the enterprise. *Commun. ACM*, 51(9):72–79, Sept. 2008.

[11] L. E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[12] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2009.

[13] B. Cate, G. Fontaine, and P. G. Kolaitis. On the data complexity of consistent query answering. *Theor. Comp. Sys.*, 57(4):843–891, Nov. 2015.

[14] J. Chomicki, J. Marcinkowski, and S. Staworko. Computing consistent query answers using conflict hypergraphs. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, pages 417–426, New York, NY, USA, 2004. ACM.

[15] 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*, pages 841–844, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[16] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[17] E. F. Codd. Relational completeness of data base sublanguages. *Research Report / RJ / IBM / San Jose, California*, RJ987, 1972.

[18] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[19] J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In J. Lee, editor, *Principles and Practice of Constraint Programming – CP 2011*, pages 225–239, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[20] A. A. Dixit and P. G. Kolaitis. A SAT-based system for consistent query answering. In M. Janota and I. Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 117–135. Springer, 2019.

[21] A. A. Dixit and P. G. Kolaitis. A SAT-based system for consistent query answering. *CoRR*, abs/1905.02828, 2019.

[22] A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: Efficient management of inconsistent databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 155–166, New York, NY, USA, 2005. ACM.

[23] A. Fuxman, D. Fuxman, and R. J. Miller. ConQuer: A system for efficient querying over inconsistent databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 1354–1357. VLDB Endowment, 2005.

[24] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, June 2007.

[25] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. on Knowl. and Data Eng.*, 15(6):1389–1408, Nov. 2003.

[26] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Found. Trends databases*, 5(4):281–393, Oct. 2015.

[27] P. G. Kolaitis and E. Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, Jan. 2012.

[28] P. G. Kolaitis, E. Pema, and W. Tan. Efficient querying of inconsistent databases with binary integer programming. *PVLDB*, 6(6):397–408, 2013.

[29] P. Koutris and J. Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '15, pages 17–29, New York, NY, USA, 2015. ACM.

[30] P. Koutris and J. Wijsen. Consistent query answering for primary keys. *SIGMOD Rec.*, 45(1):15–22, June 2016.

[31] P. Koutris and J. Wijsen. Consistent query answering for self-join-free conjunctive queries under primary key constraints. *ACM Trans. Database Syst.*, 42(2):9:1–9:45, June 2017.

[32] M. Manna, F. Ricca, and G. Terracina. Consistent query answering via ASP from different perspectives: Theory and practice. *CoRR*, abs/1107.4570, 2011.

[33] M. C. Marileo and L. E. Bertossi. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data Knowl. Eng.*, 69(6):545–572, 2010.

[34] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11):1190–1201, Aug. 2017.

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

[36] J. Wijsen. Consistent query answering under primary keys: A characterization of tractable queries. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 42–52, New York, NY, USA, 2009. ACM.

[37] J. Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 179–190, New York, NY, USA, 2010. ACM.

[38] J. Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 37(2):9:1–9:35, June 2012.

[39] J. Wijsen. Charting the tractability frontier of certain conjunctive query answering. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '13, pages 189–200, New York, NY, USA, 2013. ACM.