CAvSAT: Answering Aggregation Queries over Inconsistent Databases via SAT Solving

Akhil A. Dixit
University of California Santa Cruz
USA

Phokion G. Kolaitis
University of California Santa Cruz and IBM Research
USA

ABSTRACT

Consistent Query Answering (CQA) is a rigorous and principled approach to answering queries posed against inconsistent databases. Computing consistent answers to a Select-Project-Join (SPJ) query or an SPJ query with aggregation operators on a given inconsistent database can be an intractable problem. We demonstrate CAvSAT, a system for CQA that leverages a set of natural reductions from a given CQA instance to boolean Satisfiability (SAT) and its optimization variants. CAvSAT is the first system that is capable of handling unions of SPJ queries with aggregation operators SUM and COUNT, and databases that are inconsistent w.r.t. key constraints, functional dependencies, and denial constraints.

CCS CONCEPTS

• **Theory of computation → Incomplete, inconsistent, and uncertain databases**.

KEYWORDS

consistent query answering, SAT solving, range consistent answers

ACM Reference Format:
Akhil A. Dixit and Phokion G. Kolaitis. 2021. CAvSAT: Answering Aggregation Queries over Inconsistent Databases via SAT Solving. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3448016.3452749

1 INTRODUCTION

A relational database instance is *inconsistent* if it violates one or more integrity constraints, such as primary keys or functional dependencies, on its schema. Managing inconsistencies in real-world databases is important because poor data quality is estimated to cost the US economy alone over 3 trillion dollars each year [1]. Data cleaning is the main approach used in industry towards managing inconsistent databases (see the survey [18]). Data cleaning is often ad hoc since the user often has to make arbitrary choices while cleaning the data; for example, if a person has two different social security numbers in a database, which of the two should be kept? The framework of database repairs and consistent query answering, introduced by Arenas, Bertossi, and Chomicki [2], is an alternative, and arguably more principled, approach to data cleaning. In contrast to data cleaning, the inconsistent database is left as is; instead, 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 main algorithmic problem in this framework is to compute the consistent answers to a query q on a given database $\mathcal{I}$, that is, the tuples that lie in the intersection of the results of q applied on each repair of $\mathcal{I}$ (see the monograph [6]). Computing the consistent answers to a query q on $\mathcal{I}$ can be computationally harder than evaluating q on $\mathcal{I}$, because an inconsistent database instance exponentially many repairs. In particular, computing the consistent answers of a fixed Select-Project-Join (SPJ) query can be a coNP-complete problem. Frequently asked database queries often involve one of the standard aggregation operators MIN(A), MAX(A), SUM(A), COUNT(A), COUNT(*), or AVG(A). Such queries are called *aggregation queries* and they may also involve a GROUP BY clause that partitions a relation into disjoint groups so that the aggregate operators are applied separately to each group. Thus, an aggregation query is of the form

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

where $f(A)$ is an aggregation operator 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? An aggregation query may return different answers on different repairs of an inconsistent database; thus, there is typically *no* consistent answer as per the earlier definition of consistent answer. To obtain meaningful semantics to aggregation queries, Arenas et al. [3] introduced the *range consistent answers*.

Let Q be a scalar aggregation query. The set of *possible answers* to Q on a database instance $\mathcal{I}$ inconsistent w.r.t. a fixed set Σ of integrity constraints consists of the answers to Q over all repairs of $\mathcal{I}$ w.r.t. Σ, i.e., $\text{Poss}(Q, \mathcal{I}, \Sigma) = \{Q(\mathcal{J}) \mid \mathcal{J} \text{ is a repair of } \mathcal{I} \text{ w.r.t. } \Sigma\}$. By definition, 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 $\text{Poss}(Q, \mathcal{I}, \Sigma)$ of possible answers to Q on $\mathcal{I}$. For example, the range consistent answers to the query

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

on the instance in Table 1 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. [4] focused on scalar aggregation queries only. Fuxman, Fazli, and Miller [14] extended the notion of range consistent answers to aggregation queries with grouping. For a query

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452749

Table 1: Running example – an inconsistent database instance $\mathcal{I}$ (primary key attributes are underlined)

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

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

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

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

we say that a tuple $(T, [glb, lub])$ is a *range consistent answer* to Q on $\mathcal{I}$, if the following three conditions hold: (i) For every repair $\mathcal{J}$ of $\mathcal{I}$, there exists d s.t. $(T, d) \in Q(\mathcal{J})$ and $glb \leq d \leq lub$; (ii) For some repair $\mathcal{J}$ of $\mathcal{I}$, we have that $(T, glb) \in Q(\mathcal{J})$; (iii) For some repair $\mathcal{J}$ of $\mathcal{I}$, we have that $(T, lub) \in Q(\mathcal{J})$.

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

Several academic prototype systems for consistent query answering have been developed [4, 5, 8, 10, 13–15, 17, 19, 21, 22]. These systems use different approaches, including as logic programming [5, 17], compact representations of repairs [7], or reductions to solvers [11, 13, 19, 21]. Among these, only the ConQuer system [14, 15] is capable of handling aggregation queries with grouping. However, ConQuer can only handle an aggregation query with grouping, provided the underlying SPJ query belongs to a class called C_{forest}. For such a query Q, the range consistent answers of Q are SQL-rewritable, i.e., there is a 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 with grouping, including all queries whose range consistent answers are not SQL-rewritable or are NP-hard to compute. Up to now, no system supports such queries.

Here, we demonstrate CAvSAT (Consistent Answers via Satisfiability Solving), the first system to compute the range consistent answers to all aggregation queries involving the aggregation operators `SUM(`A`)`, `COUNT(`A`)`, `COUNT(*)` with or without grouping. Note that, while an earlier version of CAvSAT supporting the consistent answers to SPJ queries (but not to aggregation queries) was reported in [10, 11], this is the first time that CAvSAT is demonstrated. A full account of CAvSAT with technical results concerning the range semantics and a detailed experimental evaluation are in [12].

The distinctive feature of CAvSAT is that it uses natural reductions to reduce the consistent answers of SPJ queries and the range consistent answers of aggregation queries to Boolean satisfiability (SAT) and to optimization variants of SAT, such as Partial MaxSAT and Weighted Partial MaxSAT. It then deploys powerful SAT solvers, such as the MaxHS solver [9], to compute the answers. Moreover, CAvSAT can handle databases that are inconsistent not only w.r.t. primary keys but w.r.t. arbitrary *denial* constraints, a much broader class of constraints that contains primary keys and functional dependencies as special cases [6].

2 CAVSAT SYSTEM OVERVIEW

2.1 System Architecture

CAvSAT has a modular architecture, which is shown in Figure 1.

Query Pre-processor Module This module takes the query and the set of integrity constraints as inputs and attempts to determine the complexity of computing the consistent answers to the query. For SPJ queries without self-joins and with primary key constraints, the module implements the *attack graph* algorithm [20] to determine whether or not the consistent answers are SQL-rewritable. If the consistent answers are SQL-rewritable, the query is forwarded to the Query Re-writing module. If the consistent answers are not SQL-rewritable or their complexity is not known in the pre-processing stage, the queries are forwarded to one of the SAT-solving modules depending on whether or not the query has aggregate operators.

Query Re-writing Module This module implements the query-rewriting algorithms in [14, 15, 20]. For queries with SQL-rewritable consistent answers, this module produces the consistent rewriting and evaluates it on the inconsistent database instance directly to obtain the consistent answers to the original input query.

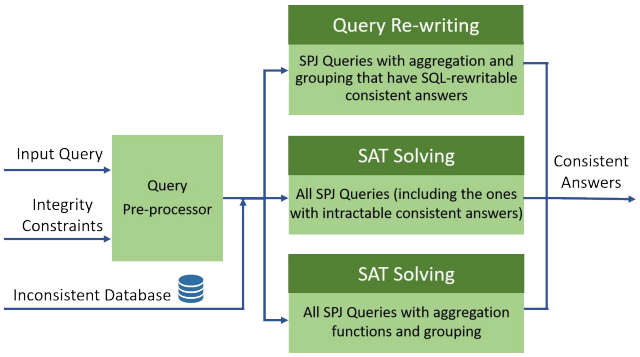

Figure 1: Modular architecture of CAvSAT

SAT Solving Modules These two modules reduce a given CQA instance to an instance of Boolean satisfiability (SAT) or one of its variants via polynomial-time reductions. After constructing such a SAT instance, these modules invoke a SAT solver to compute an (optimal) solution and then extract the consistent answers to the query. For queries without aggregation, the reductions and the detailed workings of the SAT-solving module for queries are presented in our earlier paper [11]. Here, we focus on demonstrating the SAT-solving module tailored to answer queries with the aggregation operators `SUM(`A`)`, `COUNT(`A`)`, and `COUNT(`$*$`)`. In the SAT-solving module for aggregation queries, we reduce the computation of the range consistent answers to Partial MaxSAT for `COUNT(`$*$`)` and `COUNT(`A`)`, and to Weighted Partial MaxSAT for `SUM(`A`)`. Due to space limitation, we only present the reduction for the `SUM(A)` aggregation operator.

2.2 Workflow of the SAT Solving Module

Let $Q :=$ `SELECT SUM(`A`) FROM` $T(U, A)$ be an aggregation query where $T(U, A)$ is a relation expressed using an SPJ query q. Let $\mathcal{I}$ be a database instance and $\mathcal{G}$ be the set of groups of tuples of $\mathcal{I}$ that share the values of the key attributes. Moreover, let $\mathcal{W} = W_1, \cdots, W_m$ be the set such that each W_i is a minimal set of tuples that satisfy the query q^* on $\mathcal{I}$, where $q^* :=$ `SELECT 1 FROM` $T(U, A)$.

Reduction 2.1. *For each tuple f_i of $\mathcal{I}$, introduce a boolean variable x_i. Construct a weighted partial CNF formula ϕ 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 tuples in G_j such that $m \neq n$, construct a hard clause $\alpha_j^{mn} := (\neg x_m \vee \neg x_n)$.*

(2) 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 $w_j = ||q^*(W_j)||$, 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 $W_j \in \mathcal{W}_N$, introduce a new variable y_j and let

$$\beta_j = (y_j, w_j) \text{ and } \gamma_j = \left(\left(\bigvee_{f_i \in W_j} \neg x_i\right) \vee y_j\right) \wedge \left(\bigwedge_{f_i \in W_j} (\neg y_j \vee x_i)\right).$$

Construct a weighted partial CNF formula

$$\phi = \left(\bigwedge_{j=1}^{|\mathcal{G}|} \alpha_j\right) \wedge \left(\bigwedge_{j=1}^{|\mathcal{G}|} \left(\bigwedge_{\substack{f_m \in \mathcal{G}_j \\ f_n \in \mathcal{G}_j}} \alpha_j^{mn}\right)\right) \wedge \left(\bigwedge_{j=1}^{|\mathcal{W}|} \beta_j\right) \wedge \left(\bigwedge_{W_j \in \mathcal{W}_N} \gamma_j\right).$$

It can be shown that if $\mathcal{I}$ is a database instance, Q is an aggregation query with `SUM(`A`)`, and ϕ is the WPMaxSAT-instance constructed using Reduction 2.1, then, in a maximum (a minimum) satisfying assignment of ϕ, the sum of weights of the falsified clauses is the glb-answer (lub-answer) in the range consistent answers $\textsc{Cons}(Q)$ on $\mathcal{I}$. We illustrate the workflow of computing range consistent answers using Reduction 2.1 using an example. Let $\mathcal{I}$ be the database instance from Table 1 and let Q be the following aggregation query Q, which asks for the sum of all account balances belonging to a customer named Mary:

```
SELECT SUM(ACCT.BAL) FROM CUSTOMER, ACCT, CUSTACCT
WHERE CUSTOMER.CID = CUSTACCT.CID AND ACCT.ACCID =
CUSTACCT.ACCID AND CUSTOMER.CNAME = 'Mary'
```

From Reduction 2.1, 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: $(\neg x_2 \vee \neg x_7, 1000), (\neg x_3 \vee \neg x_7, 1000), (\neg x_2 \vee \neg x_8, 1200), (\neg x_3 \vee \neg x_8, 1200), (y_1, 100), (y_2, 100)$;
γ-clauses: $(\neg x_2 \vee \neg x_9 \vee y_1), (\neg y_1 \vee x_2), (\neg y_1 \vee x_9), (\neg x_3 \vee \neg x_9 \vee y_2), (\neg y_2 \vee x_3), (\neg y_2 \vee x_9)$.

Observe that the the variables corresponding to the tuples in the `CUSTACCT` relation can be omitted, because `CUSTACCT` does not violate Σ. The witnesses $\{f_2, f_9, f_{13}\}$ and $\{f_3, f_9, f_{13}\}$ belong to $\mathcal{W}_N$ due to the account balance being -100 in both cases, so we introduce new variables y_1 and y_2 respectively, and construct hard γ-clauses as described above. The β-clauses corresponding to these witnesses are $(y_1, 100)$ and $(y_2, 100)$ respectively. Observe that the sum of weights of all soft clauses of ϕ is 4400. An assignment in which $x_8 = 0$ and $x_9 = 1$ is a maximum satisfying assignment to the PMaxSAT instance ϕ constructed from above clauses. The sum of satisfied soft clauses by this assignment is 3500 since it satisfies two clauses with weights 1200 each, one with weight 1000 and one with weight 100. Thus, the glb-answer is 4400 - 3500 = 900. Similarly, setting $x_8 = 1$ and $x_9 = 0$ yields a minimum satisfying assignment in which the sum of satisfied soft clauses is 2200, since it satisfies one clause with weight 1200 and one with weight 1000, indicating that the lub-answer is 4400 - 2200 = 2200. Hence, we have that $\textsc{Cons}(Q, \mathcal{I}) = [900, 2200]$.

The preceding Reduction 2.1 assumes that the constraints on the schema consist entirely of keys and, actually, that there is one key per relation. The reduction can be extended to arbitrary denial constraints on the schema - the details can be found in [12].

3 IMPLEMENTATION OVERVIEW

Figure 2 depicts the implementation of CAvSAT at a high level. We use the Microsoft SQL Server 2019 to store the inconsistent database instance accompanied with a set of integrity constraints. Since we need the database server to store inconsistent data and handle inconsistencies at query time, the integrity constraints are not applied on the database instance within the server, but are stored separately as a relation. The reductions from consistent answers to variants of SAT are implemented in Java 8, and are hosted as a Spring application on a web server. This server also runs MaxHS, a powerful Weighted MaxSAT solver [9]. The web-based graphical user interface of CAvSAT is implemented using the ReactJS front-end development framework that connects to the CAvSAT back-end via REST API calls. The CAvSAT source code is available at the GitHub repository https://github.com/uccross/cavsat via a BSD-style open-source license.

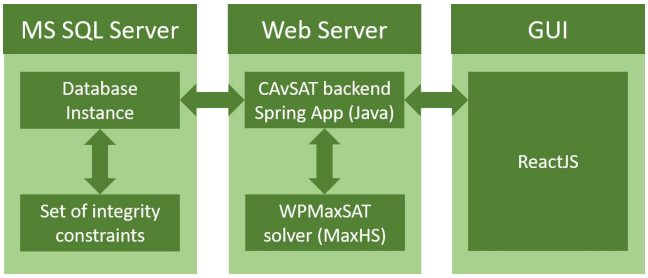

Figure 2: Implementation overview of CAvSAT

From our experimental evaluation, we observed that the reductions that encode the given CQA instance into a suitable variant of SAT take the majority of the overall time taken by CAvSAT to compute the range consistent answers to aggregation queries. This is because the SAT-solving modules need to run SQL queries on the inconsistent data in order to construct the clauses of the CNF formula and it is a time-consuming task if the input query has more joins or high selectivity. We have tested queries involving joins with up to six relations over large databases, but it remains to be seen whether constructing the clauses corresponding to the minimal witnesses is a computational bottleneck causing the performance of CAvSAT to deteriorate as the number of joins in the query increase.

4 DEMONSTRATION OVERVIEW

We describe the features of the system that we will demonstrate. First, the user connects to a remote database server via SQL server

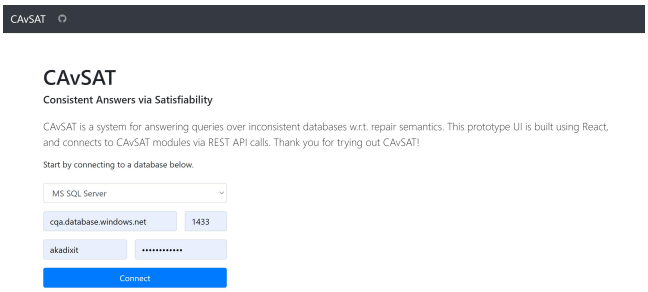

Figure 3: CAvSAT GUI while connecting to a remote database using SQL server authentication

authentication, as shown in Figure 3. After a connection is established, the user is asked to choose the database instance on which queries will be evaluated. The top right corner of the user interface in Figure 4 shows that the user is connected to a Microsoft SQL Server at localhost and ready to run queries on a database instance named `bank_accounts`. CAvSAT allows the user to specify the input query either using SQL or with first-order logic syntax. The user has a choice between evaluation strategies for computing consistent answers: the default is to use SAT solving, while two SQL-rewriting algorithms from [14, 20] can be used for SPJ queries having SQL-rewritable consistent answers. After evaluating a query, CAvSAT internally runs a suitable solver (or evaluates an SQL-rewriting) and displays the consistent answers (or range consistent answers in case of an aggregation query) to the user (Figure 4).

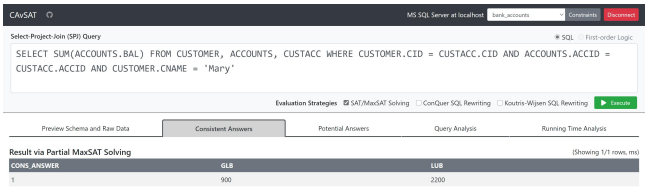

Figure 4: CAvSAT GUI after computing the range consistent answers to an aggregation query from Section 2.2

CAvSAT has several other features, including the following:

(a) The user can preview the selected schema, the integrity constraints on the schema and the raw data in the tab titled *Preview Schema and Raw Data*.

(b) The *Potential Answers* tab displays the answers to the input query evaluated directly on the inconsistent database instance. This not only helps users to visualize the difference between potential and consistent answers, but to also determine the overhead of computing consistent answers compared to evaluating the input query on the database directly.

(c) The *Query Analysis* tab shows the complexity of computing consistent answers to the query if it is determined by the Query Pre-processor module. The attack graph and the join graph of the input query are also visualized (Figure 5).

(d) The *Running Time Analysis* tab shows the break-down of the internal tasks performed by CAvSAT while computing the consistent answers along with the time taken to complete each of those tasks (Figure 6).

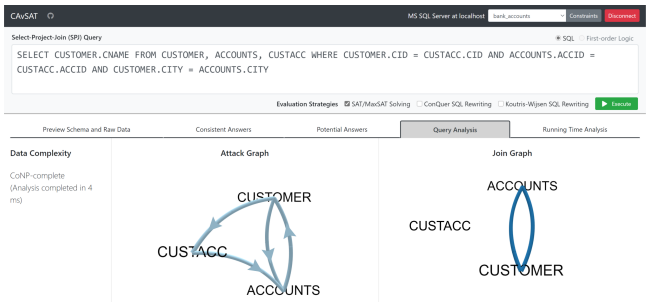

Figure 5: Query Analysis tab showing an attack graph of a query: a visual explanation on why computing consistent answers to the input query is coNP-complete

SAT / Partial MaxSAT Solving

Attribute	Value
Time to compute answers from the consistent part of the database (ms)	18
Time to compute minimal witnesses to the query (ms)	107
Time to compute relevant facts (ms)	27
Time to attach FactIDs to the relevant facts (ms)	26
Time to create positive clauses from key-equal groups (ms)	4
Time to create negative clauses from minimal witnesses (ms)	60
Time to write the clauses to a DIMAC file (ms)	3
Time to eliminate inconsistent potential answers (ms)	125
Total SAT-solving time (ms)	69
Time to write the final consistent answers to a table (ms)	75
Total Evaluation Time (ms)	498

Figure 6: Screenshot of the break-down of time taken to compute consistent answers using the SAT-solving module

An extensive experimental evaluation of CAvSAT is presented in [12]. This evaluation involves both synthetic and real-word databases, and a variety of aggregation queries with and without grouping. The first set of experiments included a comparative study of CAvSAT with ConQuer [14, 16]. In a standalone performance evaluation of CAvSAT, we experimented with TPC-H databases of varying sizes starting from 0.5 GB (8 million tuples) to up to 5 GB (44 million tuples) and varying degree of inconsistency w.r.t. primary key constraints from 5% to 35%. The experiments with a real-world database were carried out to evaluate CAvSAT's performance in presence of functional dependencies and denial constraints. We obtained promising results from both sets of experiments that corroborate the applicability and the scalability of CAvSAT.

Acknowledgments Akhil Dixit is supported by a Baskin School of Engineering Dissertation-Year Fellowship and by the Center for Research in Open Source Software (CROSS) at the University of California Santa Cruz.

REFERENCES

[1] The four V's of big data. https://www.ibmbigdatahub.com/infographic/four-vs-big-data.

[2] 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.

[3] 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.

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

[5] P. Barceló and L. E. Bertossi. Logic programs for querying inconsistent databases. In *Practical Aspects of Declarative Languages, 5th Int. Symposium, PADL 2003, New Orleans, LA, USA, January 13-14, 2003, Proceedings*, pages 208–222, 2003.

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

[7] 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.

[8] 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.

[9] 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.

[10] A. A. Dixit. CAvSAT: A system for query answering over inconsistent databases. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1823–1825, New York, NY, USA, 2019. Association for Computing Machinery.

[11] 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 Int. Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proc.*, volume 11628 of *Lecture Notes in Computer Science*, pages 117–135. Springer, 2019.

[12] A. A. Dixit and P. G. Kolaitis. Consistent answers of aggregation queries using SAT solvers. *CoRR*, abs/2103.03314, 2021.

[13] S. Flesca, F. Furfaro, and F. Parisi. Querying and repairing inconsistent numerical databases. *ACM Trans. Database Syst.*, 35(2):14:1–14:50, 2010.

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

[15] 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.

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

[17] 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.

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

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

[20] 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.

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

[22] 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.