A SAT-Based System for Consistent Query Answering

Amrutham Lakshmi Himaja (A20474105)

Atharva Kadam (A20467229)

Saikrishna Rachha (A20471757)

Shivcharan Bharathi (A20496930)

Abstract

Consistent Query Answering (CQA) is a principled and rigorous approach to the semantics of queries given against inconsistent databases. The intersection of the responses to a query on every repair, i.e. on every consistent database that differs from the supplied inconsistent one in a minimal way, is the consistent answers to a query on an inconsistent database. Even if any fixed conjunctive query is efficiently computable on a consistent database, computing the consistent responses of a fixed conjunctive query on an inconsistent database can be a coNP-hard problem.The interval [glb, lub] of the greatest lower bound and the least upper bound such that the answer to the question on every repair falls within the range of consistent answers to an aggregating query. It can be NP-hard to compute the range of consistent replies to a fixed aggregate query.

The query is first-order rewritable without aggregation operators. CAvSAT, the first SAT-based system for consistent query responding, was designed, constructed, and assessed. CAvSAT makes use of a collection of natural reductions derived from the challenge of calculating consistent answers to Boolean Satisfiability variations. The system can handle arbitrary denial constraints and unions of conjunctive queries, including functional dependencies as a particular instance. Wide range of tests are conducted on both synthetic and real-world databases to illustrate CAvSAT's use and scalability.

Introduction

An inconsistent database is one that breaches one or more integrity constraints, such as key restrictions or functional dependencies.. In the real world, inconsistent databases occur for a variety of causes and in a variety of scenarios, including data warehousing and information integration, where dealing with inconsistency is a major difficulty. The collection of data from heterogeneous sources where the data sources follow their own integrity requirements. Real-world databases becoming inconsistent could also be due to database engines inability to handle complicated integrity constraints.

The SAT solution to construct a system for consistent query response that can handle a wide range of practical inquiries and integrity requirements in this thesis. SAT (Boolean Satisfiability) is a classic NP-complete problem, and there has been a lot of research into various elements of satisfiability and SAT solving. SAT solvers have also been used to resolve open problems in mathematics.

Main aim is to manage the inconsistency of the database and find an apt approach for it. To address breaches of integrity requirements in a given inconsistent database, data cleaning, clustering techniques, and/or domain knowledge are applied, resulting in a single consistent database.Arenas, Bertossi, and Chomicki- framework of database repairs introduced consistent query answering Several systems have been proposed in the past but this paper proposes a CAvSAT system (Consistent answers via SAT).

Problem Statement

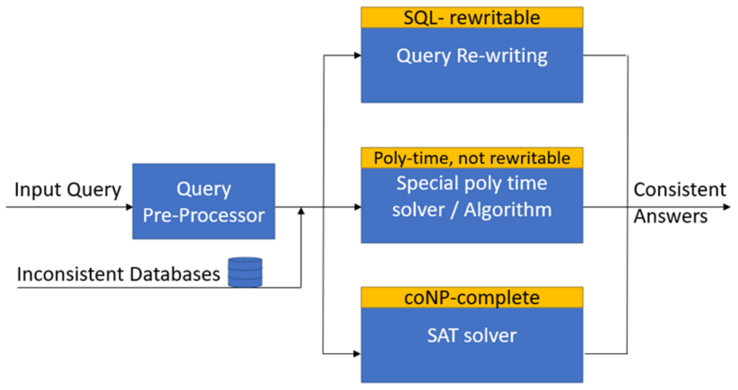
Experiments testing CAvSAT on both synthetic and real-world datasets are presented, demonstrating its use and scalability. The first set of tests focused on providing consistent replies to conjunctive queries under key constraints on synthetic databases with up to one million tuples per relation. The second series of studies focused on the consistency of replies to (unions of) conjunctive queries with functional dependencies.

Even if every fixed conjunctive query is efficiently computable on a given consistent database, computing the consistent replies of a fixed conjunctive query on a given inconsistent database can be a coNP-hard task., a SAT-based method for consistent query responding. Unions of conjunctive questions and arbitrary denial restrictions may be handled by the system, which includes functional dependencies as a specific instance.

Methodology

CAvSAT – "Consistent Answers via SAT Solving", a complete and scalable SAT-based system capable of computing consistent answers, is introduced in this paper.

CAvSAT System Architecture:
The architecture of CAvSAT, takes advantage of extensive research on the difficulty of calculating consistent replies to a range of query classes and integrity criteria. The following sections explain how each of the modules works. The architecture of CAvSAT shown in below figure.



Modular Architecture of CAvSAT

1. Query Preprocessor Module:
   The Query Preprocessor module takes the query and a set of database schema integrity constraints as inputs and tries to figure out how difficult it is to compute consistent replies to the

query. This module initially verifies whether the input query belongs to the class C_{forest} or the class $C_{aggforest}$ if the database schema only includes primary key constraints [6].

If it does, the question is sent to the Query Rewriting module, which generates a SQL query (i.e., SQL-rewriting based on ConQuer [6] algorithms) that computes the consistent or range consistent replies to the original input query. If the consistent replies are SQL-rewritable, the query is transmitted to the Query Rewriting module, which uses Koutris and Wijsen's rewriting methods to derive the SQL query and compute the consistent responses.

Because of the presence of self-joins, aggregation operators, or integrity requirements on the schema that are beyond primary keys, the consistent responses are not SQL-rewritable or their complexity is unknown in the pre-processing stage. Depending on whether the question contains aggregate operators, it is directed to one of the SAT-solving modules.

2. Query Rewriting Module:

   The query-rewriting algorithms are implemented in this module. This module generates the consistent rewriting and evaluates it on the inconsistent database instance directly to acquire the consistent answers to the original input query for queries having SQL-rewritable consistent replies.

   Consistent rewritings based on ConQuer algorithms are built directly in SQL, but rewritings based on Koutris and Wijsen algorithms are built first in tuple-relational calculus and then translated into SQL to evaluate in a typical database engine.

3. SAT Solving Modules:

These two modules use polynomial-time reductions to reduce a given CQA instance to an instance of Boolean satisfiability (SAT) or one of its variants. The DIMACS file format, which is one of the common ways to express CNF formulas, is used to store the SAT instances. These modules use a suitable state-of-the-art SAT solver to compute an optimal solution after creating an instance of SAT or one of its variants.

The SAT-solving module, for example, uses the Glucose solver [7] or the CaDiCaL [8] solver to answer Boolean SAT instances, whereas the MaxHS solver [9] is used to solve Partial MaxSAT or Weighted Partial MaxSAT cases. If an optimal assignment is identified, the solver uploads it to a file, which CAvSAT reads to decode the information about the range of consistent solutions to the question.

CAvSAT reads a solution output by a solver, updates the SAT instance in the DIMACS file if needed, then calls the solver again for the following iteration in the case of iterative SAT techniques. Because most recent solvers can read the DIMACS file format, CAvSAT can readily accommodate a new, more efficient solver if one is produced in the future.

Inconsistent Databases:

The inconsistent database, unlike data cleansing, is not cleaned; instead, inconsistencies are addressed at query time by examining all possible database repairs. The main algorithmic challenge in this system is to compute consistent replies to a query on a particular database. Computing consistent responses to a database query might be computationally more difficult than evaluating because an inconsistent database may have exponentially many fixes.

There have been several academic prototype systems for consistent query responding developed. The system employs answer set programming reductions, whereas the EQUIP system employs binary integer programming reductions and the subsequent deployment of

CPLEX. However, it is fair to conclude that there is currently no complete and scalable system for consistent query answering; this situation has hampered the widespread acceptance of the framework of repairs and consistent responses as a principled alternative to data cleansing.

Basically, managing database inconsistencies is an ancient but persistent issue. A database that breaches certain integrity constraints is said to be inconsistent. Some explanations for database inconsistency include: all ICs are not enforced by DBMS, data from many databases is combined, on an already existing database, new constraints are placed, constraints imposed by the user. We don't want to repair the database to restore consistency in numerous cases: there isn't any permission, it's too costly, inconsistency that is just momentary. Inconsistency has been researched extensively in a variety of circumstances. In classical logic, an inconsistency of formulae implies that each formula is incorrect (triviality). If a database instance does not comply with integrity constraints, it is said to be inconsistent (constraint violation).

The two techniques used to solve the inconsistencies in the databases (approaches used to solve them) are as follows:

1. Data cleaning:

Data Cleaning is a widely used engineering solution for dealing with inconsistent databases. The primary idea behind data cleaning is to employ simple database operations like tuple deletions, tuple insertions and tuple updates to resolve integrity constraints violations and create a single consistent version of the database that can be used against queries. Data cleaning does not ensure what data will be kept or eliminated during the procedure. The disadvantage is that cleaning up the same database in different ways yields different results for the same query.

It is the major strategy of the industry. It's more engineering than science because it's an ad hoc technique that generally necessitates arbitrary choices while cleaning the data. To remedy violations of integrity constraints, clustering techniques and/or domain knowledge are applied. The principle method to data cleaning is to use a structure of database fixes and consistent query responses. At query time, inconsistencies are dealt with by examining all possible database corrections. I is a consistent database J that is "minimally" different from I.

2. Database repairs:

A framework for coping with database inconsistencies without having to clean up dirty data. Repair is the most consistent subset of a database with inconsistencies. To obtain consistent database instances, it is critical to decide which sorts of database operations, such as tuple deletions, tuple insertions, and tuple updates, are allowed to be done on an inconsistent database instance.

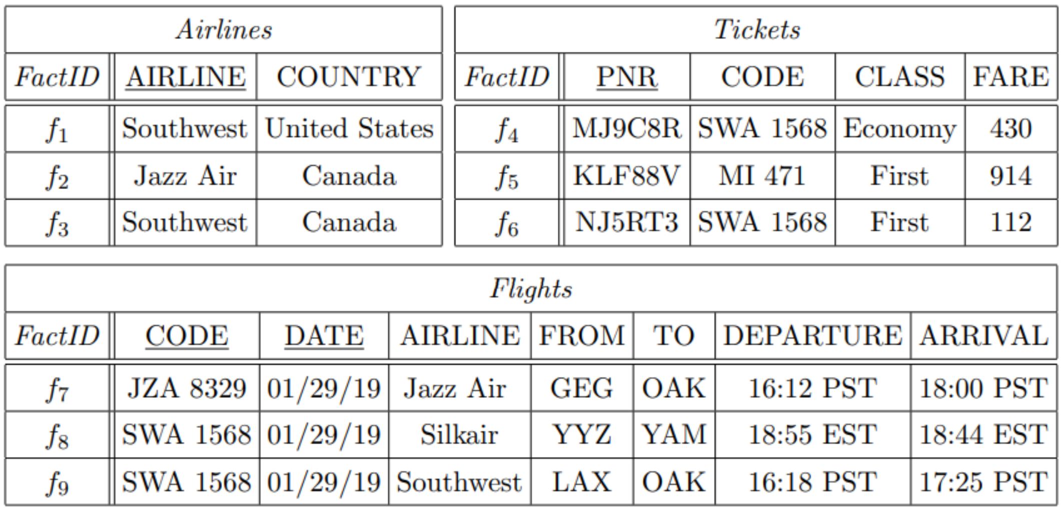
Several repair semantics have been examined based on the permissible database operations, including tuple and set-inclusion-based repairs, tuple-deletion and set-inclusion-based repairs, and tuple and cardinality-based repairs. The database repairs accomplished by conducting solely tuple-deletions on the inconsistent database instance are known as subset repairs.

Arenas, Bertossi, and Chomicki proposed a framework for database repairs and consistent query response that offers an alternative, principled, and scientific method to managing inconsistent databases. In this framework, inconsistencies are resolved during query evaluation by examining all possible database repairs, where a database repair I w.r.t. a consistent database J with "minimum" differences from I is defined by a set of integrity requirements.

I w.r.t. The consistent responses to a query q on a specific database. The intersection of the outcomes of q applied to each I repair is a set of integrity constraints. Fuxman et alConQuer's system can handle aggregation inquiries, that is, aggregation queries with key constraints for which the underlying conjunctive query, that is, the query without aggregation operators, belongs to the Cforest class. The range consistent answers of such a query Q are SQL-rewritable, which means that there is a SQL query Q′ that can be used to get the range semantic answers of Q on an instance I by directly evaluating Q′ on I.

Consistent Query Answering (CQA):

CQA (consistent query answering) is a method of querying inconsistency databases without first mending them. Arenas was the first to suggest consistent query responses. Although CQA provides a clear framework for querying inconsistent databases, computing the proportion of repairs in which a candidate answer is true, rather than simply asserting that it is true in all repairs or is false in at least one repair, is arguably more instructive.



Airlines

FactID	AIRLINE	COUNTRY
f_1	Southwest	United States
f_2	Jazz Air	Canada
f_3	Southwest	Canada

Tickets

FactID	PNR	CODE	CLASS	FARE
f_4	MJ9C8R	SWA 1568	Economy	430
f_5	KLF88V	MI 471	First	914
f_6	NJ5RT3	SWA 1568	First	112

Flights

FactID	CODE	DATE	AIRLINE	FROM	TO	DEPARTURE	ARRIVAL
f_7	JZA 8329	01/29/19	Jazz Air	GEG	OAK	16:12 PST	18:00 PST
f_8	SWA 1568	01/29/19	Silkair	YYZ	YAM	18:55 EST	18:44 EST
f_9	SWA 1568	01/29/19	Southwest	LAX	OAK	16:18 PST	17:25 PST

An inconsistent database instance of flight information records

Consistent Query Answering for Key Constraints:

In this part, we assume that R is a database schema with Σ as a finite number of primary key constraints, i.e., one key constraint per R relation. Unions of boolean conjunctive questions are the first thing we look at. We give a natural polynomial-time reduction from Certainty(q) to UnSAT for a fixed union q:= q1 ∪ … ∪ qk of boolean conjunctive queries q1,..., qk.

We next extend this reduction to unions of non-boolean conjunctive queries, such that the consistent answers to q may be calculated by iteratively solving Weighted MaxSAT instances for every fixed union q:= q1 ∪ ... ∪ qk of non-boolean conjunctive queries q1,..., qk. Cons(q) is not always equal to Cons(q1) ∪ . . . ∪ Cons(qk) of conjunctive queries (qk). The concepts of key-equal groups of facts and minimal witnesses to a union of conjunctive inquiries, which we will introduce next, are prominently used in the following sections.

Definition for Key-Equal Group:

Assume that “I” is an R-instance. Two facts of a relation R of I are said to be key-equal if they agree on R's key qualities. If every two facts in S are key-equal, and no fact in S is key-equal to some fact in I\S, the set S of facts of I is called a key-equal group of facts.
For instance, the database instance's key-equal groups of facts from Table :

{f1, f3}, {f2}, {f4}, {f5}, {f6}, {f7}, and {f8, f9}

Definition for Minimal Witness:
Let S be a subinstance of I and I be an R-instance. S is a minimum witness to a union q of conjunctive inquiries on I if S |= q, and we have that $S^{|} \forall = q$ for any suitable subset $S^{|}$ of S.

Proposed approach

1. **Identification of the problem**

When queries are assessed across inconsistent databases, consistent query answering (CQA) strives to provide relevant results. Such responses must undoubtedly be true in all repairs, which are consistent databases with a minimum variation from the inconsistent one. The intersection of the responses to a query on every repair, i.e., on every consistent database that varies from the supplied inconsistent one in a minimum way, is the consistent answers to a query on an inconsistent database. Even if any fixed conjunctive query is efficiently computable on a consistent database, computing the consistent responses of a fixed conjunctive query on an inconsistent database can be a coNP-hard issue.

Data inconsistency occurs when a database has various tables that deal with the same data but get it from separate sources. Data redundancy usually exacerbates inconsistency.

2. **Analyzing the problem**

CQA of conjunctive inquiries has been extensively researched . Computing Cons(q, I,) given an instance I is in coNP if is a fixed finite set of denial constraints and q is a k-ary conjunctive query, where k 1. Within coNP, certainty(q,) demonstrates a range of behaviors, even for key constraints and boolean conjunctive queries. A trichotomy theorem is the most conclusive finding to date. If q is a self-join-free (no repeated relation symbols) conjunctive query with one key constraint per relation, then certainty(q) is either SQLrewritable, in P but not SQL-rewritable, or coNP-complete, according to this trichotomy theorem. Furthermore, given such a question, a quadratic algorithm may be used to determine which of the three trichotomy scenarios applies. Whether or not this trichotomy extends to arbitrary boolean conjunctive queries and arbitrary denial restrictions is still an open question.

3. **Developed solution**

We present a comprehensive and scalable system, CAvSAT (Consistent Answers via SAT Solving), for answering queries over inconsistent databases. The distinctive feature of CAvSAT is that it uses several natural reductions from computing the consistent answers of queries to SAT and its optimization variants, and then deploys powerful SAT solvers. CAvSAT can handle a wide range of practical queries and integrity constraints, namely, unions of conjunctive queries with or without aggregation and grouping, over database schemata with arbitrary denial constraints.

We report an extensive experimental evaluation of CAvSAT. We carried out a suite of experiments on both synthetically generated databases and real-world databases, and for a variety of queries with and without aggregation and grouping.

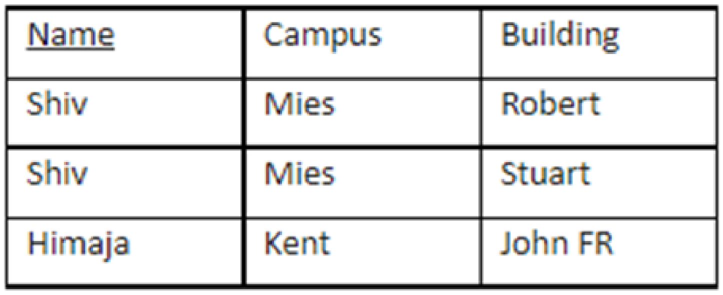
The classic and most commonly researched NP-complete issue is Boolean Satisfiability (SAT). Is a boolean formula satisfiable, as determined by the SAT? There has been a lot of research done on SAT-solving (for a summary, see), and this field of study has made remarkable progress in the previous few years (sometimes referred to as the "SAT Revolution"). SAT solvers today can answer SAT cases with millions of clauses and variables in a relatively short amount of time.

As a result of this accomplishment, SAT solvers are now widely used in industry as general-purpose problem-solving tools. Many real-world issues, such as scheduling, protocol design, software verification, model checking, and more, may be naturally expressed as SAT instances and handled fast with solvers like Glucose and CaDiCaL. A SAT solver typically accepts a boolean formula in Conjunctive Normal Form (CNF) as input and either returns a satisfactory assignment (if one exists) or informs the user that the formula is unsatisfiable. The clauses of the formula are given integer weights in an optimization variation of SAT, and the aim is to find an assignment that maximizes the sum of the weights of the fulfilled clauses. Weighted MaxSAT is the name for this form, and current solvers like MaxHS can solve Weighted MaxSAT cases quickly in reality.

Consistent query responding refers to the problem of computing consistent responses on an inconsistent database. Because they are selected from which queries, conjunctive queries (Select, Project, Join queries) are widely studied in SQL.

Take table R, for example, which contains two attributes: A and B. The first quality is key. Because the first two tuples in a database contain the same key attribute in both columns, they violate the key, suggesting that there is only one solution to this problem

In CQA, we have simple yes or no answers, referred to as boolean CQA. Although both workers have a single campus recorded in the database, Shiv has two separate buildings listed, which violates functional dependence. The first (minimal) repair is accomplished by deleting the first tuple, while the second (minimal) repair is accomplished by removing the second tuple. The query's answer is always the same.



Name	Campus	Building
Shiv	Mies	Robert
Shiv	Mies	Stuart
Himaja	Kent	John FR

Q1: SELECT * FROM Location

Because neither of the first two tuples in the query result in both repairs, the only option is to utilize the tuple (Himaja,Kent,John FR). The other question, on the other hand, is a bit more complicated.

Q2: SELECT Name, Campus FROM Location

There are two consistent answers: (Shiv,Mies) and (Shiv,Mies) since Q2 produces those two tuples in both situations (Himaja,Kent). Despite the fact that the information about Shiv's building is inconsistent, the user may utilize Q2 to derive accurate information about both workers' campus locations.

4. Findings / Results

First, we'll compare CAvSAT's performance against that of FO-rewritings on a database with primary key restrictions. We tested CAvSAT's performance on a real-world database with functional dependencies in the second scenario. CAvSAT performed as well as or better than a database engine like PostgreSQL while analyzing first-order rewritings. The results of the experiments show that using a SAT-based method can result in a complete and scalable system for consistent query responding.

Discussion / Comparison (Experimental Results)

Synthetic Data Generation:
For this set of experiments, the synthetic data were generated in two phases:

a. Generation of Consistent Data.
b. Injection of inconsistency into consistent data.

The parameters used to generate the data were:

a. number of tuples per relation (rSize)
b. degree of inconsistency (inDeg)
c. size of each key-equal group (kSize)

Generating Consistent data:
Authors have utilized a database schema with seven relations R1, R2, R3, each of arity two or three, for this set of studies. This schema has already been used to evaluate EQUIP, a query answering system that uses Integer Linear Programming to solve problems. Because each relation in the consistent database has the same amount of tuples, injecting inconsistency with the supplied kSize and inDeg will increase the total number of tuples. rSize is the number of tuples in the relation. For each of the queries utilized in the study, the data was created so that the query could be evaluated on a consistent database, in a relation with a size range of 15% to 20% of rSize.
All of the ternary relations' third attribute values were picked from a uniform distribution in the range [1, rSize/10]. This was done in order to simulate a wide number of possible responses. The remaining properties get their values from 10-character strings created at random.

Injecting Inconsistency:
The inconsistency was introduced into each relationship by inserting new tuples into the consistent data that shared the values of the key attributes with certain already existing tuples from the consistent data. Studies were conducted with inDeg levels ranging from 5% to 15%. The kSize values were evenly spread between two and five. The newly injected tuples' non-key properties were homogeneous random alphanumeric strings of length 10.

Conjunctive Queries on Synthetic Databases:
For the set of experiments on the synthetically generated databases, we used a set of 21 self-join-free conjunctive queries:

SQL-rewritable consistent answers

$$q_1(z) := \exists x, y, v, w\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, v, w))$$

$$q_2(z, w) := \exists x, y, v\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, v, w))$$

$$q_3(z) := \exists x, y, v, u, d\ (R_1(\underline{x}, y, z) \wedge R_3(\underline{y}, v) \wedge R_2(\underline{v}, u, d))$$

$$q_4(z, d) := \exists x, y, v, u\ (R_1(\underline{x}, y, z) \wedge R_3(\underline{y}, v) \wedge R_2(\underline{v}, u, d))$$

$$q_5(z) := \exists x, y, v, w\ (R_1(\underline{x}, y, z) \wedge R_4(\underline{y, v}, w))$$

$$q_6(z) := \exists x, y, x', w, d\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w) \wedge R_5(\underline{x}, y, d))$$

$$q_7(z) := \exists x, y, w, d\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, x, w) \wedge R_5(\underline{x}, y, d))$$

In P but not SQL-rewritable consistent answers

$$q_8(z, w) := \exists x, y\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, x, w))$$

$$q_9(z) := \exists x, y, w, u, d\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, x, w) \wedge R_4(\underline{y}, u, d))$$

$$q_{10}(z, w, d) := \exists x, y, u\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, x, w) \wedge R_4(\underline{y}, u, d))$$

$$q_{11}(z) := \exists x, y, w\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, x, w))$$

$$q_{12}(v, d) := \exists x, y, z, u\ (R_3(\underline{x}, y) \wedge R_6(\underline{y}, z) \wedge R_1(\underline{z}, x, d) \wedge R_4(\underline{x, u}, v))$$

$$q_{13}(v) := \exists x, y, z, u\ (R_3(\underline{x}, y) \wedge R_6(\underline{y}, z) \wedge R_7(\underline{z}, x) \wedge R_4(\underline{x, u}, v))$$

$$q_{14}(d) := \exists x, y, z, u\ (R_3(\underline{x}, y) \wedge R_6(\underline{y}, z) \wedge R_1(\underline{z}, x, d) \wedge R_7(\underline{x}, u))$$

CoNP-complete consistent answers

$$q_{15}(z) := \exists x, y, x', w\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w))$$

$$q_{16}(z, w) := \exists x, y, x'\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w))$$

$$q_{17}(z) := \exists x, y, x', w, u, d\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w) \wedge R_4(\underline{y}, u, d))$$

$$q_{18}(z, w) := \exists x, y, x', u, d\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w) \wedge R_4(\underline{y}, u, d))$$

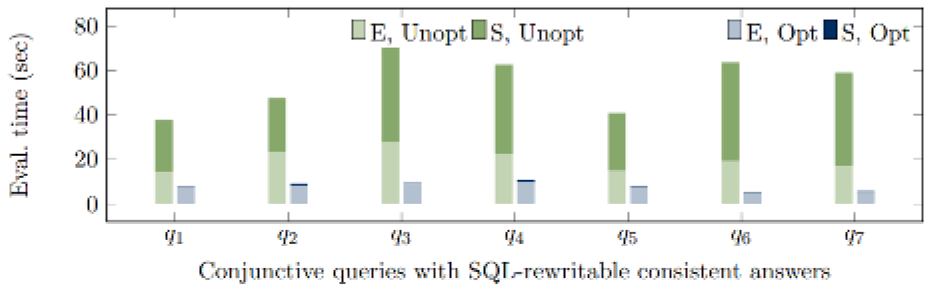
$$q_{19}(z, w, d) := \exists x, y, x', u\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w) \wedge R_4(\underline{y}, u, d))$$

$$q_{20}(z) := \exists x, y, x', w, u, d, v\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w) \wedge R_4(\underline{y}, u, d) \wedge R_3(\underline{u}, v))$$

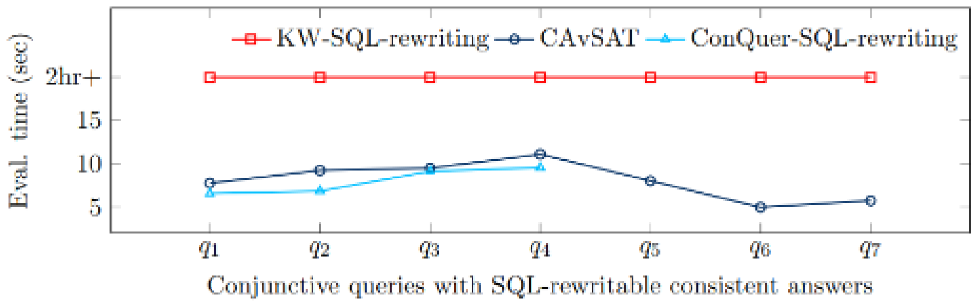
$$q_{21}(z, w) := \exists x, y, x', u, d, v\ (R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w) \wedge R_4(\underline{y}, u, d) \wedge R_3(\underline{u}, v))$$

Experimental Results on Synthetic Databases:

1. CA vs SAT on SQL- rewritable Conjunctive Queries:

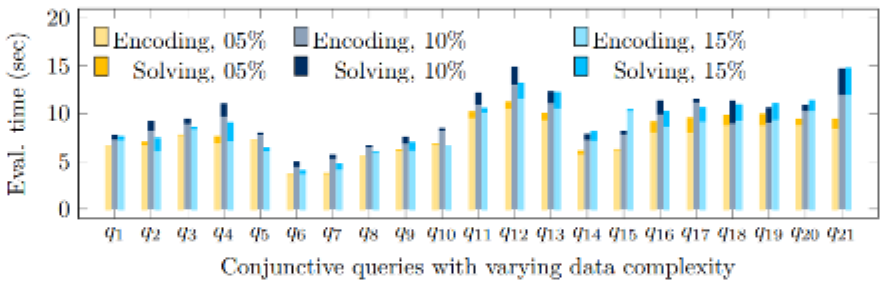


Performance of CAvSAT on conjunctive queries with SQL-rewritable consistent answers



Performance of CAvSAT, ConQuer and Koutris-Wijsen rewriting

2. CAvSAT on Harder Queries



Performance of CAvSAT on conjunctive with consistent answers of varying data complexity

3. CAvSAT on Real-world Database and Queries

We used real-world data with important limitations on each relation and one functional dependency in the next series of trials. The information pertains to food establishment inspections in Chicago and New York. Because the source did not specify the schema structure or database constraints, we split the data into four relations and assumed reasonable key constraints for all of them, as well as one additional functional dependency, as indicated in the tables below.

The schema and the constraints of the real-world database

Relation	# Tuples
NY_Insp (<u>LicenseNo</u>, Risk, <u>InspDate</u>, <u>InspType</u>, Result)	229K
NY_Rest (Name, <u>LicenseNo</u>, Cuisine, Address, Zip)	26.5K
CH_Insp (<u>LicenseNo</u>, Risk, <u>InspDate</u>, <u>InspType</u>, Result)	167K
CH_Rest (Name, <u>LicenseNo</u>, Facility, Address, Zip)	31.1K

Constraint	Type	Violations
NY_Insp (LicenseNo, InspDate, InspType → Risk, Result)	Key	25.6%
NY_Rest (LicenseNo → Name, Cuisine, Address, Zip)	Key	0%
CH_Insp (LicenseNo, InspDate, InspType → Risk, Result)	Key	0.07%
CH_Rest (LicenseNo → Name, Cuisine, Address, Zip)	Key	5.86%
CH_Rest (Name → Zip)	FD	9.73%

Performance of CAvSAT on the six queries give below:

$$q_1^r() := \exists x, y, z, w, v, y', z', w', v' \; (\text{NY\_Rest}(x, y, z, w, v) \wedge \text{CH\_Rest}(x, y', z', w', v')),$$

$$q_2^r(x) := \exists y, z, w, v, y', z', w', v' \; (\text{NY\_Rest}(x, y, z, w, v) \wedge \text{CH\_Rest}(x, y', z', w', v')),$$

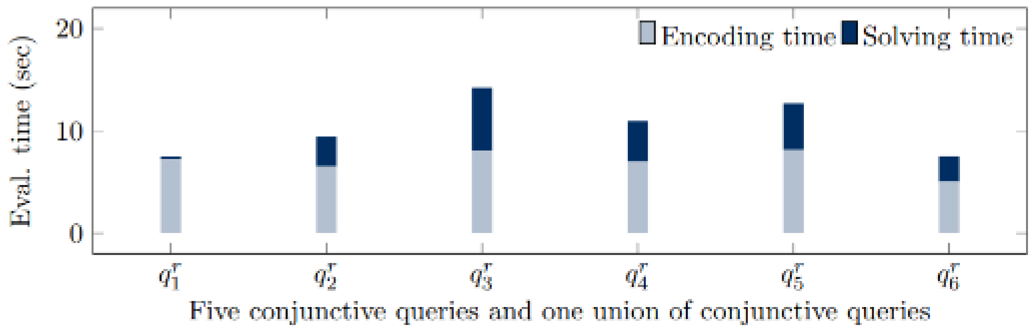
$$q_3^r(x) := \exists y, z, w, v, y', z', w', v', q, r, s, t, q', s', t' \; (\text{NY\_Rest}(x, y, z, w, v)$$
$$\wedge \text{CH\_Rest}(x, y', z', w', v') \wedge \text{NY\_Insp}(y, q, r, s, t) \wedge \text{CH\_Insp}(y', q', r, s', t')),$$

$$q_4^r(x, y) := \exists z, w, v, q, r, s \; (\text{CH\_Rest}(x, y, z, w, v) \wedge \text{CH\_Insp}(y, q, r, s, \text{'Pass'})),$$

$$q_5^r(x) := \exists y, z, w, v, q, r, s \; (\text{CH\_Rest}(x, y, z, w, v) \wedge \text{CH\_Insp}(y, q, r, s, \text{'Fail'})) \cup$$
$$\exists y, z, w, v, q, r, s \; (\text{NY\_Rest}(x, y, z, w, v) \wedge \text{NY\_Insp}(y, q, r, s, \text{'Fail'})),$$

$$q_6^r(x, v) := \exists y, z, w, y', z', w', v', q, r, s \; (\text{CH\_Rest}(x, y, z, w, v) \wedge \text{NY\_Rest}(x, y', z', w', v')$$
$$\wedge \text{NY\_Insp}(y', \text{'Not Critical'}, q, r, s)).$$

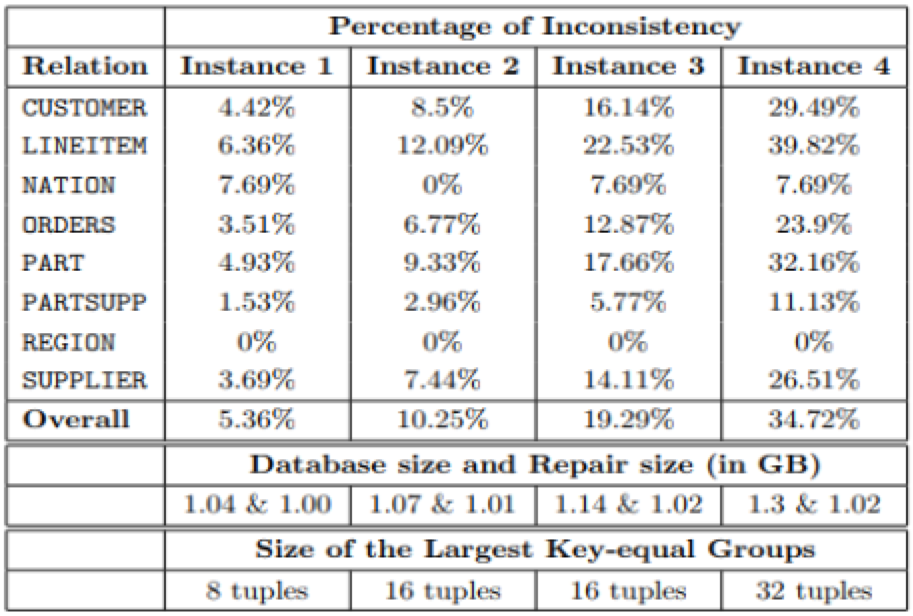
Performance of CAvSAT on the real-world food inspection database



Experiments on Aggregation Queries:

1. Experiments with TPC-H Data and Queries

Details of the TPC-H instances generated using PDBench



	Percentage of Inconsistency			
Relation	**Instance 1**	**Instance 2**	**Instance 3**	**Instance 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

2. Experimental Results on Queries without Grouping

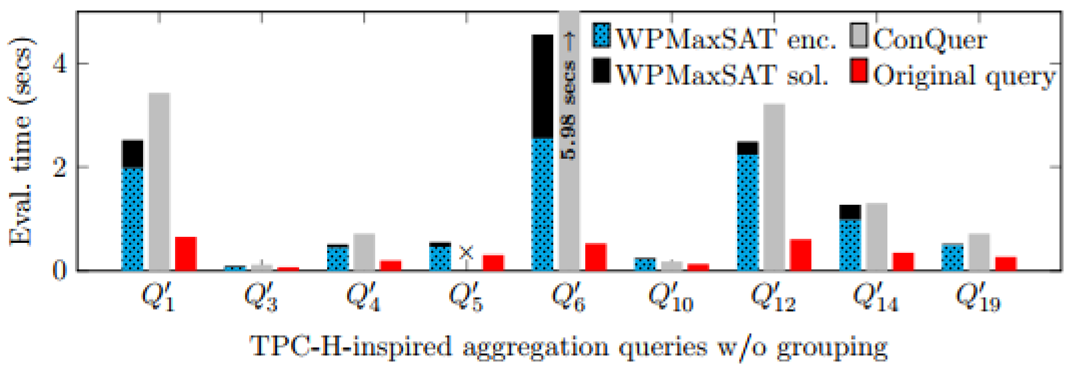
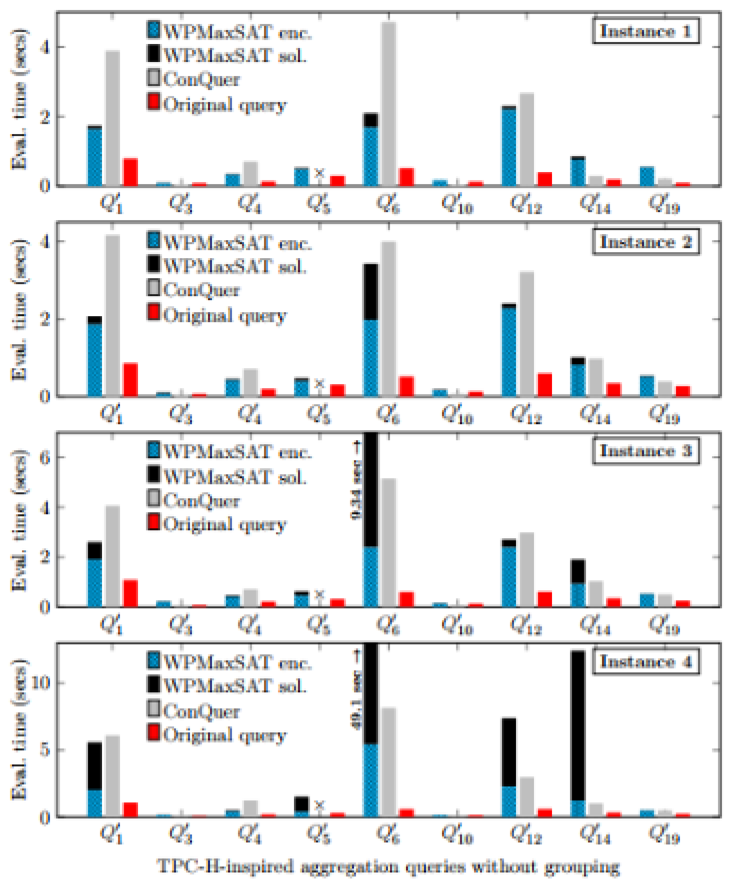


Figure 6.5: CAvSAT vs. ConQuer on TPC-H data generated using the DBGen-based tool (10% inconsistency, 1 GB repairs)



CAvSAT vs ConQuer on PDBench instances

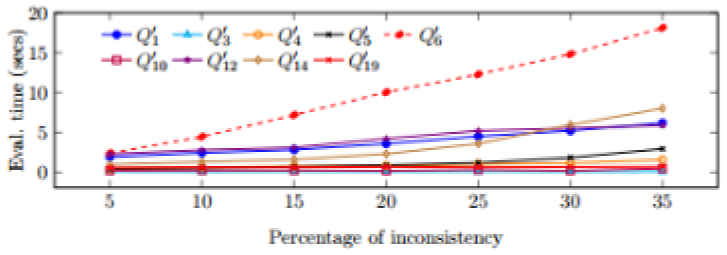
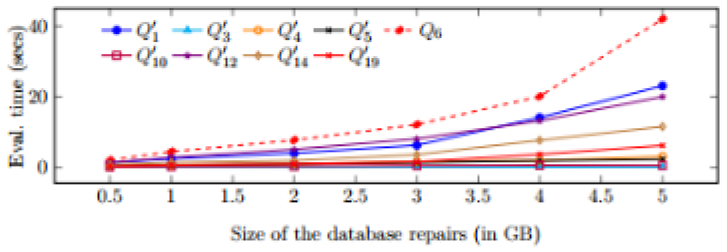


Figure 6.7: CAvSAT on TPC-H data generated using the DBGen-based tool (varying inconsistency, 1 GB repairs)



CAvSAT on TPC-H data generated using the DBGen based tool

3. Experimental Results on Queries with Grouping

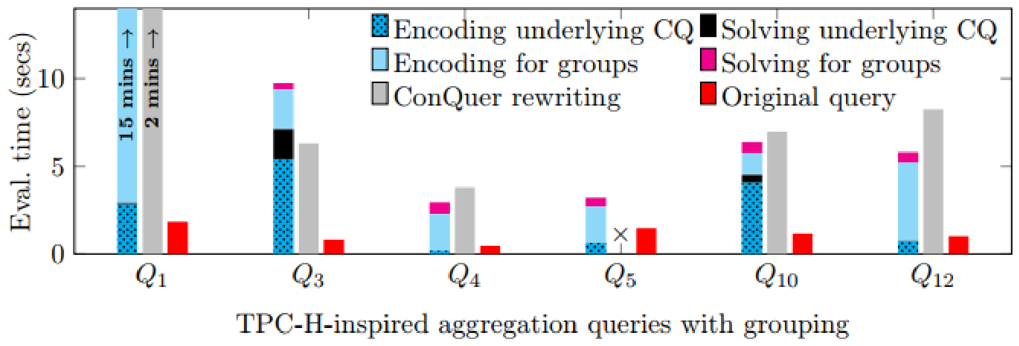
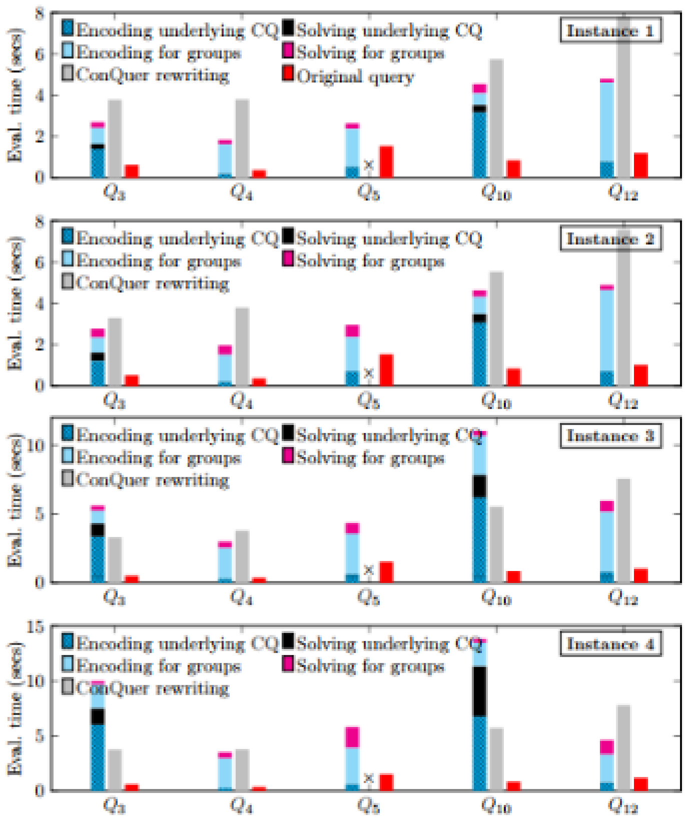
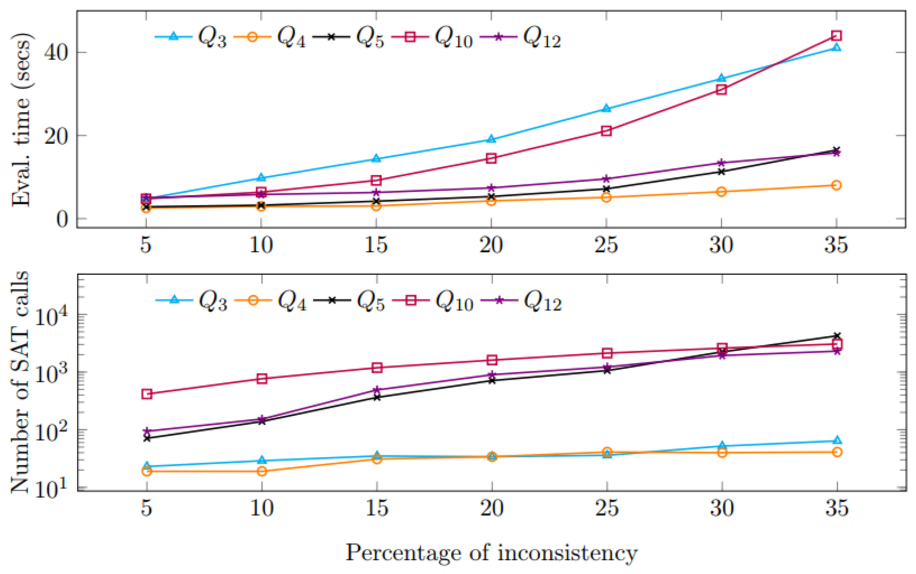
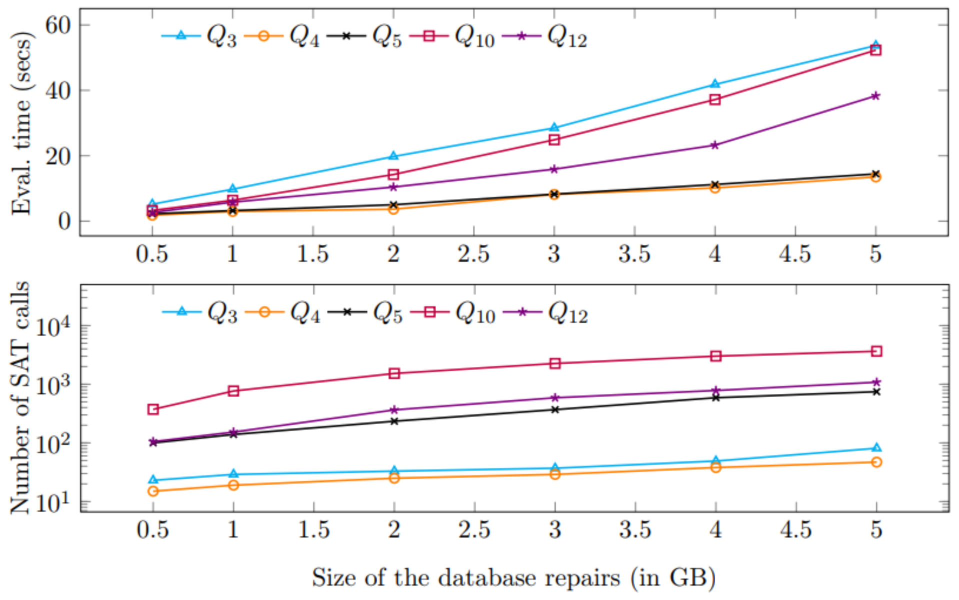


Figure 6.9: CAvSAT vs. ConQuer on TPC-H data generated using the `DBGen`-based tool (10% inconsistency, 1 GB repairs)



CAvSAT vs ConQuer on PDBench instances





4. Experiments with Real-world Data:

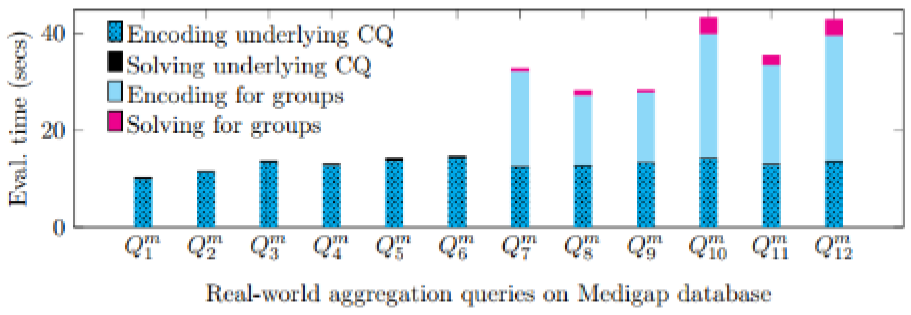
(a) Medigap schema and the size of the instance

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

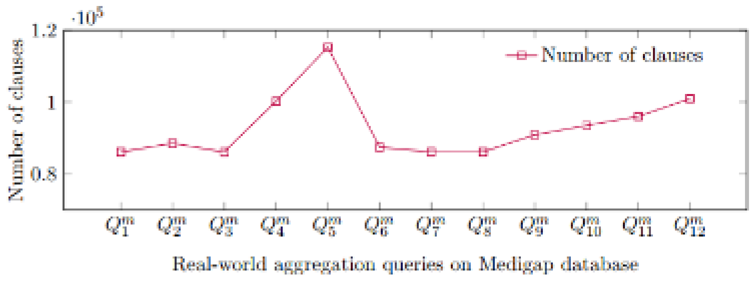
(b) Integrity constraints and inconsistency

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%

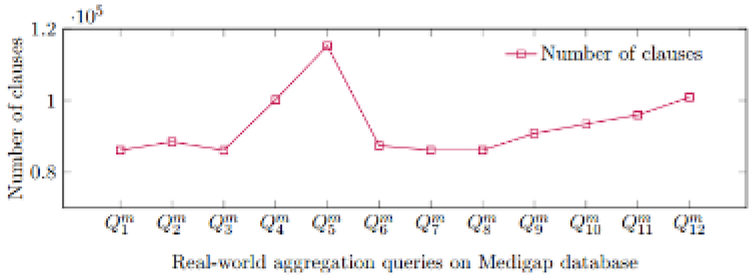
Details of the MediaGap real-world database



Performance of CAvSAT on computing the range consistent answers on a real-world database



Number of clauses in a CNF formula capturing the consistent answers to the underlying conjunctive query



Real world aggregation queries on Mediagap database

Related Work

Arenas et al. originally proposed Consistent Query Answering (CQA) as a systematic strategy for dealing with inconsistent databases. Arenas et al. defined database repairs, consistent responses, and associated challenges in their search for a strategy to deal with inconsistent databases that provides a solid semantic assurance. Cons(q, I) (or Certainty(q) for boolean queries) is the basic problem connected with consistent query answering. It is the challenge of computing consistent replies to a query q on an inconsistent database instance I. It is frequently studied from the perspectives of two complexity measures: data complexity, which is the complexity of the problem in terms of the size of the database instance I where the query q is fixed, and query complexity, which is the database instance I is fixed but the complexity is studied as a function of the size of the query q. The complexity is referred to as the combined complexity when both the database instance and the query are part of the input.

<u>The Complexity of Consistent Answers</u>

If R is a database schema, and is a fixed finite set of denial constraints on R, then the following issue is in coNP: given an R-instance I and a tuple t, is t a definite response to q on I w.r.t.? This is because we guess a repair J of I and verify that it is not a certain response to q on I w.r.t. to ensure that it is not a certain answer to q on I. (J). The previous situation prompted a series of studies on the computational complexity of computing consistent replies in order to get classification findings Researchers first focused on conjunctive query subclasses for which computing consistent replies is a tractable issue .

Under the set of integrity constraints restricted to primary keys, Fuxman and Miller identified a class of self-join-free conjunctive queries for which the consistent answers to a query q on an inconsistent database instance I are first-order rewritable (or, FO-rewritable), i.e., there exists a first-order query q 0 such that the consistent answers to q on I are precisely the result of evaluating q 0.

A specific first-order rewrite of q is a first-order query of this type. This class of inquiries is known as Ctree, and it is characterized by the concept of a query's join graph. Each atom of a self-join-free conjunctive query is represented as a node in the graph, with a directed edge from node Ri to Rj I 6= j) if any non-key variable in Ri occurs in Rj, and a self-loop on Ri if some non-key variable in Ri appears more than once in Ri.

The concept behind this approach is to apply recursion on each component of q's join graph's tree structure. The results have been expanded to include exclusion dependencies and conjunctive query unions . Koutris and Wijsen later provided the required and sufficient requirements for a query to have FO-rewritable consistent replies. Researchers have recently looked at the difficulty of consistent responses in relation to wider classes of inquiries and integrity restrictions For example, the first-order rewritability of consistent answers to self-join-free conjunctive queries on database schemata with multiple key constraints was investigated in, and consistent answers to path queries with self-joins, i.e., queries of the form x1, x2, R2(x2, x3) Rk(xk, xk+1), were investigated in. Whether a classification result like the Koutris-Wijsen trichotomy extends to arbitrary boolean conjunctive questions, as well as arbitrary functional dependencies or denial restrictions, remains an unresolved question.

The Complexity of Range Consistent Answers

For self join-free conjunctive queries, the difficulty of consistent responses has been thoroughly researched, whereas aggregation queries and range consistent answers have received very less attention. Arenas et al. presented consistent responses as an alternative to consistent replies in order to get more relevant information out of the answers to scalar aggregation queries on inconsistent databases. The authors looked at the complexity of range consistent replies to scalar aggregation questions of the type in the same study.

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

where R is a relation symbol in the database schema under consideration. The results were as follows down below

If the relation R(U, A) has just one functional dependence and f(A) is one of the aggregation operators MIN(A), MAX(A), SUM(A), COUNT(*), AVG(A), then the range of consistent replies to the query SELECT f(A) FROM R(U, A) are computed in polynomial time in the database instance's size.

Computing the range of consistent replies of the query SELECT COUNT(A) FROM R(U, A) on a given R-instance is an NP-complete problem using a database schema R consisting of a relation symbol R with one key dependence.There is a database schema R that consists of a relation symbol R with two functional dependencies, such that calculating the range of consistent replies of the query SELECT f(A) FROM R(U, A) on a given R-instance is an NP-complete issue.

Existing Systems for Consistent Query Answering

Several systems for consistent query responding have been proposed in the past, however they generally stayed as academic prototypes for a variety of reasons. The ConQuer (Consistent Querying) system is specifically designed for queries in the classes Cforest and Caggforest. Other methods employ logic programming, concise repair representations, or solver reductions. Other methods employ logic programming, concise repair representations, or solver reductions. The EQUIP system employs reductions to binary integer programming and the subsequent deployment of CPLEX, whereas the system uses reductions to answer set programming.

The ConsEx (Consistency Extractor) system was designed on the concept of disjunctive logic programming, which computes consistent responses to queries by evaluating logic programs with stable model semantics.

Comparative analysis

System	Constraints	Queries	Method
CAvSAT	Arbitrary denial Constraints	Arbitrary unions of conjunctive queries with aggregation and grouping	Reductions to SAT and its variants
ConQuer	Primary Key constraints	Subclass of conjunctive queries with aggregation and grouping	SQL-rewriting
ConsEx	Universal Constraints, Acyclic referential integrity constraints and not-null constraints	Datalog with ¬	Answer set programming
Hippo	Universal Constraints	Projection-free queries with ∪ and \	Direct algo[1]rithm
EQUIP	Primary Key Constraints	Arbitrary conjunctive query	Reductions to Binary Integer Programming

Conclusion and Recommendations

Using FO-rewritable Query Hard Queries and real-world databases, the paper compares and experiments results for CAvSAT. CAvSAT is better than using a database engine to evaluate first-order rewritings on queries with first-order rewritable consistent responses. This result reveals a potential difference between theory and practice because the examination of first-order writability of consistent responses was motivated by the need for an efficient evaluation of consistent answers using only the database engine.

To summarize the report, the framework of repairs provides a scientific approach to cope with inconsistent databases. While much progress has been made in theory, no comprehensive CQA system exists in practice. Given the advancements in SAT solving, CAvSAT seems to be a promising approach.

For future work, comparison of CAvSAT with state-of-the-art Data Cleaning systems such as holoclean.

References

1. Food Inspections, City of Chicago (Aug 2011), https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5
2. New York City Restaurant Inspection Results, Department of Health and Mental Hygiene (DOHMH) (Aug 2014), https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j

3. Arenas, M., Bertossi, L., Chomicki, J.: Consistent query answers in inconsistent databases. In: Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 68–79. PODS '99, ACM, New York, NY,USA(1999).https://doi.org/10.1145/303976.303983,http://doi.acm.org/10.1145/303976.303983

4. Arenas, M., Bertossi, L.E., Chomicki, J.: Answer sets for consistent query answering in inconsistent databases. TPLP 3(4-5), 393–424 (2003). https://doi.org/10.1017/S1471068403001832,https://doi.org/10.1017/S1471068403001832

5. Barcelo, P., Bertossi, L.E.: 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. pp. 208–222 (2003). https://doi.org/10.1007/3-540-36388-2 15, https://doi.org/10.1007/3-540-36388-2_15

6. Ariel Fuxman, Elham Fazli, and Ren´ee J. Miller. ConQuer: Efficient management of inconsistent databases. In Proc. of the 2005 ACM SIGMOD International Conference on Management of Data, SIGMOD '05, pages 155–166, New York, NY, USA, 2005. ACM.

7. Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In Proc. of the 21st International Jont Conference on Artifical Intelligence, IJCAI'09, page 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

8. CaDiCaL simplified satisfiability solver. http://fmv.jku.at/cadical/

9. Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Principles and Practice of Constraint Programming – CP 2011, pages 225–239, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.