On the Tractability and Intractability of Consistent Conjunctive Query Answering

Enela Pema
University of California, Santa Cruz
epema@soe.ucsc.edu
Advisors: Prof. Phokion G. Kolaitis, Prof. Wang-Chiew Tan

ABSTRACT

The consistent query answering framework has received considerable attention since it was first introduced as an alternative to coping with inconsistent databases. The framework was defined based on two notions: repairs and consistent query answers. Informally, a repair is a consistent database that minimally differs from the inconsistent database. The consistent answers to a query are those tuples that appear in the intersection of the answer sets of the query when evaluated over all possible repairs. Here we study the complexity of the problem of consistent query answering for the class of acyclic conjunctive queries without self-joins, under primary key constraints. The problem is known to be coNP-complete in general for this class. Our goal is to determine the boundary between tractability and intractability, by establishing a dichotomy to the effect that, every conjunctive query in this class is either in PTIME or coNP-complete. In the PTIME direction, previous work has identified the queries for which consistent answers can be computed via first-order rewriting. In fact, for the class of acyclic conjunctive queries without self-joins, under primary key constraints, the boundary between first-order rewritable and not first-order rewritable queries has already been determined. Hence, our focus is on queries for which there is no first-order rewriting. We present a technique for computing in polynomial time the consistent query answers to several not first-order rewritable queries. We hope this technique may lay the foundations for a more general algorithm that handles all PTIME not first-order rewritable queries. In the hardness direction, we identify several representative queries of the class, for which we show that the problem is coNP-hard.

This work is done under the supervision of Prof. Phokion G. Kolaitis and Prof. Wang-Chiew Tan.

Keywords
Inconsistent databases, consistent query answering, database repairs

1. INTRODUCTION

Integrity constraints are designed to preserve integrity of the data in a database. A database is consistent if the data satisfies the integrity constraints, and it is inconsistent otherwise. In enterprises, important decisions are made under the assumption that data is consistent. Nevertheless, it is common for databases to become inconsistent. One reason is that integrity constraints may not be enforced in a database. The database research community has continuously shown interest in inconsistent databases, both from the theoretical and practical point of view. One of the most common approaches developed for handling inconsistencies is data cleaning. A data cleaning process aims to completely remove violations and bring the database to a consistent state. It changes and removes information permanently in order to restore consistency. The main difficulty in data cleaning resides in having to make deterministic decisions about what tuples to keep and what to remove from the database. There are many scenarios in which there is no obvious reason for differentiating among “good” and “bad” data, and therefore, data cleaning may not be a natural approach to address the problem. For example, in a data integration setting, inconsistencies may arise because independent sources obey different integrity constraints. Therefore, when data is integrated from different sources, it may fail to satisfy the integrity constraints of the global schema.

Consistent Query Answering Framework Another alternative has been proposed for managing inconsistent data, which is to allow conflicts to occur in the database, and to return at query time, only the query answers that are ‘consistent’. The consistent query answering framework was introduced by Arenas, Bertossi and Chomicki [2]. The framework, inspired by the possible worlds semantics, defines two important notions: the notion of repairs, and the notion of consistent query answers. A repair is a consistent database which minimally differs from the inconsistent database. The consistent query answers are the certain answers of the query over all possible repairs. Two fundamental algorithmic problems have been defined: repair checking, i.e., given two instances \( r \) and \( I \) over the same schema, check if \( r \) is a repair of \( I \); and consistent query answering, i.e., given a database \( I \), check if a Boolean query \( q \) is true in every repair of \( I \).

The complexity of both these problems is data complexity, since the integrity constraints and the query are fixed. There are many possible ways of repairing a database (deleting tuples, inserting tuples, etc.). Several notions of repairs have been proposed. One of the most generally considered type
of repair is the subset repair. Subset repairs are repairs that are subsets of the inconsistent database. They are obtained when only repairing by deletions is allowed. The notion of minimality here is that no tuples are removed unnecessarily from the database while repairing. Here we consider only subset repairs and refer the user to [3] for an overview of various proposed notions of repairs. From now on, we will avoid specifying the type of the repair, and whenever we use the term ‘repair’, we will mean ‘subset repair’. Example 1 illustrates the notion of repairs and consistent query answers.

Example 1. Let Emp(ssn, name, salary) be a given relation schema and let the attribute Emp.ssn be a key. Let I = \{Emp(112, John, 20K), Emp(112, John, 30K), Emp(412, Anna, 34K), Emp(412, Ann, 34K)\}. The tuple Emp(112, John, 20K) conflicts with the tuple Emp(112, John, 30K). Also, the tuple Emp(412, Anna, 34K) conflicts with the tuple Employees(412, Ann, 34K). The instance r_1 = \{Emp(112, John, 20K), Emp(412, Ann, 34K)\} is one possible repair. Another possible repair is the instance r_2 = \{Emp(112, John, 20K), Emp(412, Ann, 34K)\}. The query q = \exists x, z. Emp(x, “Anna”, z) is not true in every repair because “Anna” does not appear at least in repair r_2.

The space of possible repairs grows exponentially with the size of the database. Thus, the problem of computing consistent query answers to a query over an inconsistent database is in general difficult. The complexity of the problem has become subject of extensive investigation aiming to delineate the boundary of tractability vs. intractability by considering these parameters: type of repairs, class of queries, and class of integrity constraints.

Earlier Work A systematic study of both problems, repair checking and consistent query answering, was carried out by Chomicki and Marcinkowski [4]. All results in [4] are given for subset repairs only. The repair checking problem was further investigated in [1]. In [4], they show that the complexity of computing consistent query answers, in general, can be as high as \Pi_2^p. However, certain restrictions on the form of constraints and queries may yield tractable subclasses. Since then, the research community has put considerable effort into identifying classes of conjunctive queries, for which the problem is in PTIME. A few approaches have been proposed to efficiently compute consistent query answers. Two of the main approaches are query rewriting and the conflict-hypergraph technique.

The basic idea of the query rewriting approach is to rewrite a given query q into a first-order query q’, such that, when evaluated over the inconsistent database, the query q’ returns exactly the consistent answers to q. However, the query rewriting technique is not powerful enough to handle all PTIME queries, since a first-order rewriting is not always possible. This technique has been studied at some depth by Fuxman and Miller [11]. They studied conjunctive queries without self-joins, under primary key constraints. They identified a class of first-order rewritable queries which they called C_{forest}, and built Conquer [10], a system that produces a first-order rewriting for any query in C_{forest}. First-order rewritability has been further studied by Wijsen [16]. He also focused on conjunctive queries that do not contain self-joins, but in particular, he looked at acyclic conjunctive queries, which are conjunctive queries that can be described by a join tree [13]. Wijsen’s main result was a necessary and sufficient condition for first-order rewritability of acyclic conjunctive queries without self-joins, under primary key constraints. He also gave an algorithm for first-order query rewriting.

The conflict-hypergraph technique was proposed and studied by Chomicki, Marcinkowski and Staworko in [4, 5]. They used a hypergraph, which they called conflict-hypergraph, to represent violations of denial constraints in a database, and they gave an algorithm which used this graph to compute consistent query answers to queries without projections, under denial constraints. They also developed a system, Hippo [6], which implements the idea of their algorithm. The conflict-hypergraph technique is limited in that it cannot be used to compute consistent answers to queries without projections.

The aforementioned results, provide sufficient conditions for tractability of consistent query answering for certain classes. However, the boundary between tractability and intractability of the problem for any significant subclass of queries and constraints, remains unclear. Some work in the direction of establishing a boundary between tractability and intractability, was performed by Fuxman and Miller [11]. They look at conjunctive queries without self-joins, under primary key constraints. The problem has been proved to be coNP-complete in general for this class [4]. Fuxman and Miller claimed a dichotomy for a specialized class of conjunctive queries without self-joins, which they call C^*.

Later on, however, Wijsen [17] showed that their claim was incorrect. For this, he gave a polynomial-time algorithm for computing the consistent query answers to the query q_2 = \exists x, y. R(x, y) \land S(y, z), claimed to be coNP-complete in [11]. This recent result brings two problems into the picture. First, it suggests that there is a potentially wide class of tractable queries, interesting in practice, but for which there is no first-order rewriting, and there is no known generalized algorithm to compute consistent answers. Second, the existence of a dichotomy for any significant subclass of conjunctive queries and key constraints, remains an open question, and to our opinion, a question deserving attention.

Research Goal Our goal is to investigate the boundary of tractability vs. intractability of consistent query answering and establish a dichotomy for some significant subclass of conjunctive queries and primary key constraints. Given that for the class of acyclic queries without self-joins and primary key constraints, we know the boundary between first-order rewritable and not first-order rewritable queries [16], we investigate the possibility of dichotomizing this class into tractable and intractable cases. A possible dichotomy for tractability vs. intractability, when combined with Wijsen’s dichotomy for first-order rewritability vs. non-rewritability, gives rise to a trichotomy for this class. The trichotomy would partition the class of acyclic conjunctive queries without out self-joins, into three parts: queries that are first-order rewritable, queries that are tractable but not first-order rewritable, and queries that are coNP-complete. Primary key constraints are an important class of constraints that are commonly used in practice. The class of acyclic conjunctive queries is a broad class of queries, well-known for their good properties. Many hard problems in databases, can be solved in polynomial time for acyclic conjunctive queries. We believe that a dichotomy on tractability vs. intractabil-
ity would be an interesting result from a theoretical point of view, and at the same time important in the practical aspect. As a theoretical result, it would prove that computing consistent query answers for acyclic conjunctive queries without self-joins, under primary key constraints, is either tractable or coNP-complete. From a practical point of view, a dichotomy, would provide a criterion for the optimization of consistent query answering. Assuming that the separating criterion can be decided in polynomial time, the dichotomy could be used from a query engine to decide what approach to take when computing consistent query answers: rewrite the query, use some other PTIME algorithm, or apply some heuristic for the coNP-hard cases. Completing the dichotomy, requires exhibiting a new algorithm for computing the consistent answers to PTIME not first-order rewritable queries, which is a terrain very little explored so far.

**Summary of Results** In this paper, we present the progress we have made towards the goal of determining the separating line between tractability and intractability for the class of acyclic conjunctive queries without self-joins and integrity constraints consisting of primary key constraints. The problem is not trivial to solve, even if we consider only queries with two atoms. In the direction of tractability, we focus on queries that are in PTIME but not first-order rewritable. We introduce a new approach for computing consistent query answers to a subclass of queries with two atoms. Our technique uses a new graph, named conflict-join graph, which represents conflicts in the inconsistent database, and pairs of tuples that join to satisfy the query. The conflict-join graph was inspired from the conflict-hypergraph. We observe that given a Boolean conjunctive query is consistently true in a database if and only if, in the conflict-join graph, the maximum cardinality of all vertex-independent sets has a particular size. The maximum independent set problem is known to be NP-complete in general. But there are classes of graphs for which the problem can be solved in polynomial time. One such class is the class of claw-free graphs. A graph is claw-free if it does not contain a claw as an induced subgraph, where a claw is the complete bipartite graph $K_{1,3}$. The class of claw-free graphs is an important class of graphs. It is a generalization of line graphs. They have gained considerable attention over time because of several desirable characteristics they have. We refer the reader to [7] for a survey on claw-free graphs. One of the key discoveries about claw-free graphs is that the problem of finding a maximum independent set can be solved in polynomial time. An algorithm to solve this problem was given by Minty [14] in 1980. There are queries whose conflict-join graph is claw-free for every database. We show that for queries that satisfy this property, we can compute consistent query answers in polynomial time. After this, we show that for a subclass of conjunctive queries with two atoms, the conflict-join graph is always claw-free. Our contribution consists in that our technique provides an alternative PTIME algorithm for computing the consistent answers to $q_1 = \exists x. y. R(x, y) \land S(y, x)$, and in addition, it allows for further expansion of the class of queries known to be in PTIME but not first-order rewritable.

In the hardness direction, we consider a new variant of Monotone 3SAT that we call Monotone 3SAT-1NEG, in which each variable has exactly one negative occurrence. This variant has been shown to be NP-complete [8]. We use this variant to prove coNP-hardness of various queries with two atoms.

## 2. PRELIMINARIES

We assume we have a fixed relational database schema which we typically denote by $P$, that is a finite collection of relation symbols, each with an associated arity. If $R$ is a relation symbol of $P$ and $r$ is an instance over $P$, then $R^r$ denotes the interpretation of $R$ on $r$. We assume a set of integrity constraints is defined over the schema and we denote it by $F$. Here we consider only key functional dependencies which can be defined as follows: Given a set $F$ of functional dependencies and a relation $R$ with list of attributes $Attr(R)$ over the schema $P$, a key constraint of $R$ is a minimal set $X$ such that $F$ entails the functional dependency $X \to Attr(R)$. Here we consider the class of conjunctive queries, which are first-order formulas that may contain only conjunctions of positive literals and existential quantification. They can be expressed in the form: $q(x) = \exists w. R_1(x_1, y_1) \land \ldots \land R_n(x_m, y_m)$, where the variables of $x_1, y_1, \ldots, x_m, y_m$ appear in exactly one of $z$ and $w$. Here, whenever we write a conjunctive query, we underline variables that appear in the positions of attributes that participate in a key constraint. For instance, when we write $q_1 = \exists x, y. R(x, y) \land S(y, z)$, we imply that the first attributes of $R$ and $S$ are key constraints in the respective relations. Variable $x$ is called a key variable of atom $R(x, y)$. In general, when a conjunctive query is presented in this form, we omit explicity specifying the schema and the set $F$ of constraints as we can derive them from the formulation of the query itself. Let $q$ be a conjunctive query and $R(x, y)$ an atom in it. We define $\text{vars}(R(x, y))$ as the set of variables appearing in the atom $R(x, y)$, i.e., $x \cup y$; and $\text{key}(R(x, y))$ as the set of key variables appearing in the atom $R(x, y)$, i.e., $x$. To simplify things, we accept more relaxed notations. Given a conjunctive query, we will refer to the atoms with the name of the corresponding relations. So, we write $\text{vars}(R)$ instead of $\text{vars}(R(x, y))$, and $\text{key}(R)$ instead of $\text{key}(R(x, y))$. We formally define subset repairs in Definition 1.

**Definition 1 (Subset Repair).** Let $I$ be a database instance, and $F$ be a set of integrity constraints. An instance $r$ is a subset repair of $I$ w.r.t. $F$ if $r$ is a maximal sub-instance of $I$ that satisfies $F$.

We now define the notion of consistent query answers based on the definition of repairs.

**Definition 2 (Consistent Query Answers).** Given a schema $P$, a set of integrity constraints $F$, an inconsistent instance $I$ over $P$, and a query $q$ over $P$, we say that a tuple $t$ is a consistent answer for $q$ if for every repair $r$ w.r.t $F$, we have $t \in q(r)$. It is inconsistent otherwise. If $q$ is Boolean, we say that $q$ is consistently true in $I$ iff for every repair $r$ of $I$ w.r.t $F$ we have $r \models q$. It is false otherwise.

If $q$ is a Boolean query we use the notation $f \models^F q$ to mean that $q$ is true in every repair w.r.t. $F$, and the notation $f \not\models^F q$ to mean that $q$ is false in at least some repair.

Under key constraints, the problem of computing consistent query answers is in coNP, for every query.

A conflict is a minimal set of facts that witnesses a violation of an integrity constraint. When the integrity constraints consist of key constraints only, conflicts are pairs
of facts. Let $R(A, B)$ be a given relation in which $A$ is a key. Let $I = \{R(a, b), R(a, b')\}$ be an instance over $R$. The set $\{R(a, b), R(a, b')\}$ is a conflict because the two tuples are enough to witness a violation of the key constraint in $R$. The two tuples $R(a, b)$ and $R(a, b')$ are called key-equal. The constant (in general the list of constants) that appears in a tuple in the position of the key, is called a key value of $R$. In our case, constant $a$ is a key value in $R$.

3. TRACTABILITY RESULTS

We present here our tractability results for conjunctive queries with two atoms under primary key constraints\(^1\). So far we know that there are queries with two atoms that are first-order rewritable, and queries with two atoms that are not first-order rewritable but are tractable. Wijnen [16] has characterized rewritable queries. What remains to characterize, are tractable but not rewritable queries. In general, a very few cases of tractable but not first-order rewritable queries have been detected. The query $\exists x, y, z. R(x, y) \land R(x, z) \land x \neq y$ is one such case. Fuxman proves in his thesis [9] that for this query there is no first-order rewriting, and then gives a polynomial-time algorithm for computing the consistent answers. This is a conjunctive query that contains an inequality. Another PTIME not first-order rewritable query is the query $\exists x, y. R(x, y) \land R(y, a)$ identified in [15]. This conjunctive query does not contain inequalities but it contains a self-join. The query $q_1 = \exists x, y. R(x, y) \land S(x, y)$ is a conjunctive query without self-joins that is also proved to be in PTIME but not first-order rewritable [17]. We propose a new graph, namely, the conflict-join graph, and use it to compute in polynomial time the consistent query answers to several Boolean queries that are not first-order rewritable, including the query $q_1 = \exists x, y. R(x, y) \land S(x, y)$. In the following section we describe our technique, and then show a class of queries for which it can be used to compute consistent answers.

3.1 Conflict-join graph

The conflict-join graph is constructed for a given Boolean conjunctive query, a given set of primary key constraints, and an inconsistent database. We define the conflict-join graph as follows:

Definition 3 (Conflict-Join graph). Let $P$ be a database schema. Let $F$ be a set of primary key constraints. Let $q$ be a Boolean conjunctive query over $P$, which contains two atoms. Let $I$ be an instance over $P$. We construct the conflict-join graph $H_{F,I,q}$ as follows:

- The set of vertices is the set of tuples in $I$.

- For every pair of tuples that form a conflict, add an edge connecting the corresponding vertices.

- For every pair of tuples that join, add an edge connecting the corresponding vertices.

The size of the conflict-join graph is polynomial in the size of the database. Let $V_0 = \{t_1, ..., t_k\}$ be a set of tuples that are key-equal. Every pair of tuples in $V_0$ forms a conflict. Therefore, $V_0$ induces a clique in the conflict-join graph. A set of key-equal tuples is maximal if it contains all key-equal tuples in the database. If $V_i$ and $V_j$ are maximal sets of key-equal tuples, it is easy to see that $V_i \cap V_j = \emptyset$. So, we can partition the set $V$ of tuples into disjoint sets $V_1, ..., V_t$ such that all vertices in a partition $V_i$ correspond to a maximal set of key-equal tuples in the database. By construction, the set of edges $E$ contains two types of edges: edges that represent conflicts, and edges that represent pairs of tuples that join. We can partition $E$ into $E_1$ and $E_2$, such that edges that represent conflicts are in $E_1$ and edges that represent joining tuples are in $E_2$.

We will make use of the notion of a maximal independent set in a graph, and later show a connection between the existence of a maximal independent set of a particular size in the conflict-join graph and the existence of a repair that does not satisfy the query.

Definition 4 (Independent set). Given a graph $G(V, E)$, a subset of vertices $M \subseteq V$ forms an independent set of vertices if the subgraph induced by $M$ contains no edges. It is a maximal independent set if no vertices can be added to it without introducing an edge. It is maximum if it is a largest cardinality independent set. We denote the size of the maximum independent set by $\alpha(G)$ and we call it the ‘independent set number’.

We are interested in $\alpha(H_{F,I,q})$. The vertices in a maximal independent set of $H_{F,I,q}(V, E)$ represent an instance that is consistent and does not satisfy the query $q$. But it is not necessarily a repair. We are interested in the existence of an independent set of a particular size.

Example 2. Let $q_1 = \exists x, y. R(x, y) \land S(x, y)$. Let $I = \{R(a, b), R(a, b'), R(a', b'), R(a', b), S(b, a), S(b, a')\}$. Figure 3.1 depicts the construction of the conflict-join graph. The set $r = \{R(a, b), S(b, a')\}$ is a maximal independent set in the graph. The instance $\{R(a, b), S(b, a')\}$ is a consistent and $q_1$ is false in it, but it is not a repair because $R(a', b)$ can be added without introducing violations. The set $r' = \{R(a, b'), R(a', b), S(b, a)\}$ is also a maximal independent set. The instance $\{R(a, b'), R(a', b), S(b, a)\}$ in this case is a repair and it does not satisfy $q_1$.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{conflict-join.pdf}
\caption{The conflict-join graph for the query and the instance in Example 2. Edges drawn as continuous lines connect tuples that form a conflict; edges drawn as dashed lines connect tuples that join.}
\end{figure}

The number of maximal key-equal groups is an upper bound on the cardinality of independent sets in the conflict-join graph. This is intuitively easy to understand. An independent set cannot contain two vertices from the same clique. So, it may contain at most one vertex per key-equal group. If the independent set number is exactly the same as...
the number of key-equal groups, then a repair in which the query is false can be constructed from an independent set of maximum cardinality. Otherwise, the query is true. We formally prove this in Lemma 1.

**Lemma 1.** Let $P$ be a database schema. Let $F$ be a set of primary key constraints. Let $q$ be a Boolean conjunctive query over $P$, which contains two atoms. Let $I$ be an inconsistent database over $P$. Let $H_{F,I,q}$ be the conflict-join graph of $F, I, q$, and let $n$ be the number of maximal key-equal groups in $I$. Then, $I \models q$ if and only if $\alpha(H_{F,I,q}) = n$.

**Proof.** Let $V$ be the set of vertices and $E$ the set of edges. As explained above, $V$ can be expressed as $V_1 \cup \ldots \cup V_n$ and $E$ expressed as $E_1 \cup E_2$. First, we show that $\alpha(H_{F,I,q}) \leq n$, where $n$ is the number of maximal key-equal groups. If there was an independent set $M$ of size greater than $n$, then it would contain two vertices from the same $V_i$. But, every two vertices from the same $V_i$ are connected. This contradicts the fact that $M$ is an independent set. Now we show that if $\alpha(H_{F,I,q}) = n$, there is a repair in which $q$ is false. Let $M$ be a maximum independent set. Let $r$ be an instance which contains all tuples whose corresponding vertex is in $M$. The instance $r$ is consistent because it contains no edges from $E_1$, and it does not satisfy $q$ because it contains no edges from $E_2$. It is also maximal because for each key-equal group, exactly one tuple is kept in $r$. Therefore, $r$ is a repair that does not satisfy $q$. In the opposite direction, let $r$ be a repair that does not satisfy $q$. Assume towards a contradiction, that $\alpha(H_{F,I,q}) < n$. Let $M$ be the set of vertices whose corresponding tuples are in $r$. We argue that $M$ is a maximal independent set. Since $r$ is consistent, no two vertexes in $M$ are connected. Since $r$ is maximal consistent, no vertex can be added to $M$ without introducing an edge. Obviously, the number of tuples in $r$ is $n$, because exactly one tuple per key-equal group must exists in the repair. Thus, $M$ is a maximal independent set of size greater than $\alpha(H_{F,I,q})$. This contradicts the definition of $\alpha(H_{F,I,q})$. □

The problem of finding a maximum independent set is NP-complete [12] in general. However, there are classes of graphs for which this problem is in PTIME. Such graphs are claw-free graphs, chordal graphs, perfect graphs, etc. If for a given query and a database, the conflict-join graph belongs to any of these classes, from Lemma 1 it follows that we can check in PTIME if the query is true in every repair. We focus in one of these classes, the claw-free graphs. In the following section we show that for a subclass of queries with two atoms, which satisfy a certain syntactic condition, the conflict-join graph is always claw-free.

### 3.2 Tractable queries

Now we have elevated the problem to the level where we can focus solely on the conflict-join graph and look into classes of graphs for which the independent set problem can be solved in PTIME. We already mentioned a few such classes. Here, we focus on claw-free graphs. A graph is claw-free if it does not contain a claw as an induced subgraph. The claw is the complete bipartite graph $K_{1,3}$ and is presented in Figure 3.2. Equivalently, we can say that a graph is claw-free if no vertex has three pairwise nonadjacent neighbors. Claw-free graphs form a broad class of graphs that enjoy good algorithmic properties. An algorithm to solve the independent set problem is given by Minty [14].

A reasonable question to ask is: "What queries are such that for every inconsistent instance the conflict-join graph is claw-free?" We identify a subclass of queries with two atoms whose conflict-join graph is always claw-free. Theorem 1 states this result.

**Theorem 1.** Let $q$ be a Boolean conjunctive query with two atoms, $R$ and $S$. Let $Q$ be the set of variables that the two atoms share, i.e., $Q = \text{vars}(R) \cap \text{vars}(S)$. If $\text{key}(R) \cup \text{key}(S) \subseteq Q$ then, for every inconsistent database $I$ the conflict-join graph $H_{F,I,q}$ is always claw-free. Hence, the problem of computing consistent query answers to $q$ is in PTIME.

**Proof.** We first make the following observations regarding the conflict-join graph of $q$:

(i) For every two adjacent edges $p(t_1, t_2), q(t_1, t_3)$ from $E_1$, there is an edge $r(t_2, t_3)$ in $E_1$. This is easy to verify. Since edges $p$ and $q$ are adjacent, the vertices $t_1, t_2, t_3$ must belong to the same partition $V_i$. We know that the subgraph induced by $V_i$ is a clique. Therefore, there must also be an edge between $t_2$ and $t_3$.

(ii) For every two edges $p(t_1, t_2), q(t_1, t_3)$ from $E_2$, there is an edge $r(t_2, t_3)$ in $E_1$. Assume $t_1$ is a tuple from $R^j$. Then, $t_2$ and $t_3$ are tuples from $S^j$. Let $t_2[x]$ denote the list of constants appearing in the positions of variables $x$. We have that $t_1[Q] = t_2[Q]$ and $t_1[Q] = t_3[Q]$. Since $\text{key}(Q) \subseteq Q$, it follows that $t_2[\text{key}(S)] = t_3[\text{key}(S)]$. Tuples $t_2$ and $t_3$ are both tuples from $S^j$ and they have the same constants appearing in the positions of the key attributes. So, $t_2$ and $t_3$ are key-equal, and therefore, there must be an edge between $t_2$ and $t_3$ to represent the conflict. If instead $t_1$ is a tuple from $S^j$, then $t_2, t_3$ are tuples from $R^j$. Because $\text{key}(R) \subseteq Q$, it follows that $t_2[\text{key}(R)] = t_3[\text{key}(R)]$. Again, there must be an edge between $t_2$ and $t_3$.

Now we prove that the conflict-join graph is claw-free. Assume towards a contradiction that there is a set of vertices $\{t_1, t_2, t_3, t_4\}$ that induces a claw. Let $K = \{(t_1, t_2), (t_1, t_3), (t_1, t_4)\}$ be the edges in this claw. In $K$, there are at least two edges both from $E_1$, or at least two edges both from $E_2$. Let $(t_1, t_2) \in E_1$ and $(t_1, t_3) \in E_1$. From (i) it follows that there must be an edge $(t_2, t_3) \in E_1$. Let $(t_1, t_2) \in E_2$ and $(t_1, t_3) \in E_2$. From (ii) it follows that there must be an edge $(t_2, t_3) \in E_1$. In either case, the subgraph induced by $\{t_1, t_2, t_3, t_4\}$ cannot be a claw.

Now we show how to check for any given inconsistent instance $I$, if $I \models q$ or not. First, we construct the conflict-join graph $H_{F,I,q}$. The size of the graph is polynomial to the size of the database, so this step runs in polynomial time. Next, we determine the independent set number $\alpha(H_{F,I,q})$ using Minty’s PTIME algorithm [14]. We compare $\alpha(H_{F,I,q})$ to the number $n$ of key-equal groups. From Lemma 1 it follows that $I \models q$ if $\alpha(H_{F,I,q}) = n$, and $I \not\models q$ if $\alpha(H_{F,I,q}) < n$. □
Theorem 1 gives a sufficient condition for tractability of queries with two atoms. There are some interesting queries that fall into this class, including the symmetric join query $q_1 = 3x, y. R(x, y) \land S(y, x)$. We show a few such queries in the following example:

**Example 3.** All queries presented in this example are not first-order rewritable according to [16]

- Let $q_1 = 3x, y. R(x, y) \land S(y, x)$. The variables that $R$ and $S$ share are $Q = \{x, y\}$. Because $\text{key}(R) \cup \text{key}(S) = \{x, y\} \subseteq Q$, it follows that the conflict-join graph is claw-free.

- Let $q_2 = 3x, y, z. R(x, y, z) \land S(y, x, z)$. Here $Q = \{x, y, z\}$, and $\text{key}(R) \cup \text{key}(S) = \{x, y\} \subseteq Q$; hence the conflict-join graph is claw-free.

- Let $q_3 = 3x, y, z. R(x, y, z) \land S(y, z, x)$. Here $Q = \{x, y, z\}$, and $\text{key}(R) \cup \text{key}(S) = \{x, y, z\} \subseteq Q$. The conflict-join graph is claw-free.

Theorem 1 provides an alternative PTIME algorithm for the query $q_1 = 3x, y. R(x, y) \land S(y, x)$. One may want to compare our algorithm with the one presented in [17]. Furthermore, it applies to a broader number of queries that are not rewritable. Some interesting queries in the class are the ones given in Example 3. We hope this technique may pave the way to a more general PTIME algorithm for computing consistent query answers of tractable not first-order rewritable conjunctive queries.

We point out that there are as well tractable queries for which the conflict-join graph may contain a claw. Such an example is the rewritable query $3x, y, z. R(x, y) \land S(y, z)$.

4. **Intractability Results**

In the intractability direction, the simplest query known so far for which the problem of computing consistent query answers is coNP-complete is the query $q_2 = 3x, x'. y. R(x, y) \land S(x', y)$. This was proven by a reduction from Monotone 3SAT [11]. Given a formula $\Phi$ in Monotone 3SAT form, an instance $I$ over the schema of $q_2$ is constructed by generating a tuple $R(\Phi, i, p)$ for every variable $p$ that occurs in a positive clause $\Phi_i$, and by generating a tuple $S(\Phi_j, p)$ for every variable $p$ that occurs in a negative clause $\Phi_j$. Then it is proved that $\Phi$ is true if and only if $q_2$ is false. This is proved showing that for every repair that makes $q_2$ false, we can construct a truth assignment for the formula and vice-versa. This reduction can be easily adapted for several other queries. But there are queries with two atoms which are hard, as we will later prove, for which a similar reduction as the one shown above does not work. One such query is $q_3 = 3x, x', y. R(x, x', y) \land S(x', y)$. A reduction from Monotone 3SAT, similar to the one shown above would generate a tuple $R(\Phi_i, \Phi_j, p)$ for every variable $p$ that occurs in a positive clause $\Phi_i$ and in a negative clause $\Phi_j$, and a tuple $S(\Phi_j, p)$ for every variable $p$ that occurs in a negative clause $\Phi_j$. Here, given a repair in which $q_3$ is false, it is not clear that we can construct a valid truth assignment for the formula. Instead, we are able to prove that the problem of computing consistent query answers for $q_3$ is coNP-complete using a new variant of Monotone 3SAT, namely, Monotone 3SAT-1NEG in which each variable has exactly one negative occurrence. This variant is NP-complete. The proof was given by Feder [8].

**Theorem 2.** The problem of computing consistent query answers to query $q_6 = 3x, x', y. R(x, x', y) \land S(x', y)$ is coNP-complete.

**Proof.** We prove this by a reduction from Monotone 3SAT-1NEG. Let $\Phi$ be an instance of Monotone-3SAT-1NEG. We construct $I$ over the schema of $q_6$ doing the following: (i) for each variable $p$, add a tuple $R(\Phi_i, \Phi_j, p)$ if $p$ occurs positively in $\Phi_i$ and negatively in $\Phi_j$, and (ii) for each variable $p$, add a tuple $S(\Phi_j, p)$ if $p$ occurs negatively in $\Phi_j$. Now we prove that $\Phi$ is satisfiable iff there exists a repair $r$ of $I$ in which $q_6$ is false.

($\Rightarrow$) Let $\theta$ be a satisfying assignment for $\Phi$. Construct an instance $r$ doing the following: (i) for each positive clause $\Phi_i$, pick a literal $p$ in it such that $\theta(p) = \text{true}$ and add a tuple $R(\Phi_i, 1, p)$ in $r$. (ii) for each negative clause $\Phi_j$, pick a literal $p$ in it such that $\theta(p) = \text{false}$ and add a tuple $S(\Phi_j, p)$ to $r$. First we show that $r$ is a repair. In $r'$, we put exactly one tuple for each positive clause. Therefore, the key of $R$ is not violated in $r$ and no more facts can be added to $r'$. Now we show that $r \neq q$. This holds because a variable $p$ cannot be evaluated to true and false at the same time. Therefore, $\exists x, y. R(x, x', y) \land S(x', y) = \text{false}$ in $r$.

($\Leftarrow$) Let $r$ be a repair of $I$ such that $r \neq q$. We construct an assignment $\theta$ for $\Phi$ doing the following: for each variable $p$, if there is a fact $R(\Phi_i, p)$ in $r$ make $\theta(p) = \text{true}$; for each variable $p$, if there is a fact $S(\Phi_j, p)$ in $r$, make $\theta(p) = \text{false}$; if no fact $R(\Phi_i, p)$ or $S(\Phi_j, p)$ exists in $r$, then assign to $p$ any value. Now we show that $\theta$ is a valid assignment and that it satisfies $\Phi$. In $r$, there cannot be two facts $R(\Phi_i, p)$ and $S(\Phi_j, p)$ at the same time. What if there are such facts? Since the query is false in $r$, then $\Phi$ must be different from $\Phi_j$. If this is the case, then variable $p$ is appearing twice in negative form, which is impossible in Monotone-3SAT-1NEG. So, the variable $p$ will be evaluated to either true or false. Now we show that $\theta$ makes $\Phi$ true. Because $r$ is a repair, for every positive clause $\Phi_i$, there is a tuple $R(\Phi_i, 1, p)$ and in $r$. Therefore, $\theta$ will satisfy at least one literal in every positive clause. Reasoning in similar way, we conclude that $\theta$ will satisfy at least one literal in every negative clause.

There are several queries for which hardness can be proven by a reduction from Monotone 3SAT-1NEG, including:

- $q_7 = 3x, x', y, w. R(x, w, x', y) \land S(x', x, y)$,

- $q_8 = 3x, y, z, w. R(x, y, z, w) \land S(x, w, y)$, etc.

5. **Concluding Remarks**

In this paper, we reported the results of our study on the tractability and intractability of the problem of computing consistent query answers to conjunctive queries under primary key constraints. We state our goal towards establishing a dichotomy, if possible, between queries that are in PTIME and queries that are coNP-complete, for the subclass of acyclic conjunctive queries without self-joins. Here, we focused on queries with only two atoms, since the dichotomy is not trivial even for such a simple class. On the direction of tractability, we looked at queries that are tractable but not first-order rewritable. We proposed the conflict-join graph as a representation of conflicts and joining tuples in a database. Later, we showed a connection between the problem of finding a maximum size independent
set in the conflict-join graph and the problem of consistent query answering. Given that the independent set problem is in general hard, we looked at classes of graphs for which this problem is known to be solvable in polynomial time. In particular, we looked at claw-free graphs. We showed that for a subclass of queries with two atoms, the conflict-join graph is always claw-free. Thus, these queries are tractable. Even though the class we have identified is small, it contains some interesting queries including $3x, y.R(x, y) \land S(y, x)$. Moreover, using our technique we can prove tractability for several other queries that are not first-order rewritable. We believe that this approach may provide some insight towards a generalized PTIME algorithm for computing consistent answers to a broader class of queries that are not first-order rewritable. The conflict-join graph can be generalized to queries with more than two atoms. In this case, the sets of tuples that join have more than two elements. Therefore, the conflict-join graph turns into a hypergraph. The connection between the independent set problem and consistent query answering problem, as stated in Lemma 1, remains valid even for the general conflict-join hypergraph. This introduces new challenges, since the independent set problem has not been studied on hypergraphs as much as on simple graphs. On the side of hardness, we identify several hard queries that have not been studied before. We prove their hardness via a reduction from a variant of Monotone 3SAT, in which the number of occurrences of every negated variable is restricted to be exactly one. We call this variant Monotone 3SAT-1NEG. We believe that this variant can be used to prove hardness for more complex queries which may contain an arbitrary number of atoms.

6. REFERENCES