New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins

Cibele Freire
Wellesley College

Neil Immerman
University of Massachusetts Amherst

Wolfgang Gatterbauer
Northeastern University

Alexandra Meliou
University of Massachusetts Amherst

ABSTRACT
The resilience of a Boolean query is the minimum number of tuples that need to be deleted from the input tables in order to make the query false. A solution to this problem immediately translates into a solution for the more widely known problem of deletion propagation with source-side effects. In this paper, we give several novel results on the hardness of the resilience problem for binary conjunctive queries with self-joins (i.e., conjunctive queries with relations of maximal arity 2) with one repeated relation. Unlike in the self-join free case, the concept of triad is not enough to fully characterize the complexity of resilience. We identify new structural properties, namely chains, confluenes and permutations, which lead to various NP-hardness results. We also give novel involved reductions to network flow to show certain cases are in P. Overall, we give a dichotomy result for the restricted setting when one relation is repeated at most 2 times, and we cover many of the cases for 3. Although restricted, our results provide important insights into the problem of self-joins that we hope can help solve the general case of all conjunctive queries with self-joins in the future.

1 INTRODUCTION
Various problems in database research, such as causality, explanations, and deletion propagation, examine how interventions in the input to a query impact the query’s output. An intervention constitutes a change (update, addition, or deletion) to the input tuples. In this paper, we study the resilience of a Boolean query with respect to tuple deletions. Resilience is a variant of deletion propagation that focuses on Boolean queries: it corresponds to the minimum number of tuples whose deletion causes the query to evaluate to false. In previous work [14], we provided a full characterization of the complexity of resilience for the family of self-join-free conjunctive queries (sj-free CQs) with functional dependencies. In this paper, we augment the previous results to account for a restricted class of self-joins.

Self-joins have long plagued the complexity study of many problems in database theory research: for example, on the topic of consistent query answering, Kolaitis and Pema [26] proved a dichotomy into PTIME and coNP-complete cases for the family of queries with only two atoms and no self-joins. Koutris and Suciu [27] extended the dichotomy to the larger class of self-join-free conjunctive queries, where each atom has as primary key either a single attribute or all the attributes. Koutris and Wijsen [29, 31] further extended the dichotomy to the full class of sj-free Boolean CQs, and queries with negated atoms [30]. To the best of our knowledge, there is no known result on this problem for a query family that permits self-joins. As another example, complexity results on the problem of query-based pricing [28] are also restricted to the class of sj-free CQs. On the closely related topic of deletion propagation with view side-effects, Kimelfeld et al. [24] used a characteristic of the query structure (head domination) to formalize a complexity dichotomy for the family of sj-free CQs, and indicated that self-joins can significantly harden approximation in the problem of deletion propagation. Extensions to the cases of functional dependencies [23] and multi-tuple deletions [25] also focused on the same query class. These examples offer strong indication that self-joins introduce significant hurdles in the study of a variety of problems, and progress in cases that account for self-joins is rare.1

In this paper, we give several novel results on the hardness of the resilience problem for CQs with self-joins. We focus on the class of binary CQs (those where each relation has arity of no more than 2) and provide various complexity results for binary CQs with a single instance of a repeated atom that is repeated maximal 2 or 3 times. For the case with at most 2 atoms, we prove that a P versus NP-complete dichotomy exists. The significance of our results lies in the inclusion of self-joins in our target query class.

Contributions and outline.
• Contrasting with current knowledge about the resilience of CQs without self-joins (summarized in Section 2), we demonstrate how self-joins complicate the problem and invalidate several aspects and intuitions from the self-join-free case (Section 3).
• We establish foundations for tackling the resilience problem for binary conjunctive queries (i.e., conjunctive queries with atoms of arity no more than 2) with self-joins by identifying important conditions on the minimality and connectedness of queries and by revising the fundamental notion of query domination (Section 4).
• We prove that resilience for queries that contain a triad (a structure that characterizes hardness in the sj-free case [14]) remains NP-complete in the presence of self-joins (Section 5.2).
• By further narrowing our target class of binary CQs to those where only one relation can be in a self-join (i.e., only one relation can appear in multiple atoms of the query), we identify a new structure that implies hardness, thus expanding the NP-complete class compared to the sj-free case (Section 6).
• We identify and define the fundamental structures of chains, confluenes, and permutations, and use them to prove a complete dichotomy between NP-complete and PTIME cases for the class of binary conjunctive queries where at most two atoms may correspond to the same relation (Section 7).

1While some prior work on related problems does allow for self-joins [1, 5, 9], the complexity characterizations in those results are not specific to the queries, but rather to high-level operators (e.g., join, projection, etc.). In contrast, our work provides results that are fine-grained and identify elements of the query structure that render the resilience problem NP-complete or PTIME-computable.
• We prove several involved results using the chains, confluences, and permutations structures in the case of binary conjunctive queries where at most 3 atoms correspond to the same relation. While a complete dichotomy for this class remains elusive, our work creates a roadmap and identifies remaining open problems (Section 8).

2 BACKGROUND AND PRIOR RESULTS
This section introduces our notation, defines the resilience of a query, and summarizes prior complexity results for sj-free queries.

Standard database notations. We use boldface to denote tuples or ordered sets, (e.g., x = (x₁, . . . , xₜ)) and use both subscripts and superscripts as indices (e.g., a¹ and aₜ). We fix a relational vocabulary R = (R₁, . . . , Rₜ), and denote ar ty(Rᵢ) the arity of a relation Rᵢ. We call unary and binary those relations with arity 1 or 2, respectively. A database instance over R is D = (R₁D, . . . , RₜD), where each RᵢD is a finite relation. We call the elements of RᵢD tuples and write Rᵢ instead of RᵢD when D is clear from the context. With some abuse of notation we also denote D as the set of all tuples, i.e. D = ∪ᵢ Rᵢ, where the union is understood to be a disjoint union (thus each tuple belongs to only one relation). The active domain dom(D) is the set of all constants occurring in D. The size of the database instance is n = |D|, i.e. the number of tuples in the database.²

A conjunctive query (CQ) is a first-order formula q(y) = ∃x (g₁(x) ∧ . . . ∧ gₘ(x)) where the variables x = (x₁, . . . , xₜ) are called existential variables, y = (y₁, . . . , yₚ) are called the head variables (or free variables), and each atom (also called subgoal) gᵢ represents a relation gᵢ ⊆ R(xᵢ) where xᵢ ⊆ x ∪ y.³ A self-join-free CQ (sj-free CQ) is one where no relation symbol occurs more than once and thus every atom represents a different relation. In turn, a self-join CQ is one where at least one relation symbol is repeated. We write var(gᵢ) for the set of variables occurring in atom gᵢ. As usual, we abbreviate a non-Boolean query in Datalog notation by q(y) := g₁, . . . , gₘ where q has head variables y and g₁, . . . , gₘ represents the body of the query. For tuple t of the same length as y, we write D |= q[t/y] to mean that t is in the query result of the query q(y) over database D. The set of such results is denoted by q(y)D.

Additional notations. We call “binary queries” those queries that contain only unary or binary relations. Unless otherwise stated, a query in this paper denotes a Boolean CQ q (i.e., y = φ). We write D |= q to denote that the query q evaluates to true over the database instance D, and D |= q to denote that q evaluates to false. For a Boolean query q, we write q(x) to indicate that x represents the set of all existentially quantified variables. We write [k] as short notation for the set {1, . . . , k}.

We call a valuation of all existential variables that is permitted by D and that makes q true (i.e. D |= q[w/x]) a witness w.³ The set of witnesses is then

\[ \text{witnesses}(D, q) = \{ w \mid D \models q[w/x] \}. \]

Since every witness implies exactly one set of at most m tuples from D that make the query true, we will slightly abuse the notation and also refer to such sets of tuples as “witnesses.” For example, consider the query qchain = R(x, y), R(y, z) with x = (x, y, z) over the database D = \{ t₁: R(1, 2), t₂: R(2, 3), t₃: R(3, 3) \}. Then one can easily see that

\[ \text{witnesses}(D, q_{\text{chain}}) = \{ (1, 2, 3), (2, 3, 3), (3, 3, 3) \} \]

and their respective tuples are \{ t₁, t₂, t₃, \} and \{ t₄ \}.

In line with prior work [14, 32], relations may be specified as exogenous, meaning that tuples from these relations cannot be deleted.³ We specify the atoms corresponding to exogenous relations with a superscript "x". The remaining atoms are endogenous. We may use the superscript "n" to emphasize that an atom is endogenous. Moreover, we can refer to a database as a partition of its tables into its exogenous and endogenous parts, D = Dⁿ ∪ Dⁿ.³

Complexity theory. For completeness, we briefly recall the notions of reduction and equivalence:

Definition 1 (Reduction and Equivalence). For two decision problems, S, T ⊆ \{0, 1\}ⁿ, we say that S is reducible to T (S ≤ T) if there is an easy-to-compute reduction f : \{0, 1\}ⁿ → \{0, 1\}ⁿ s.t.

\[ \forall w \in \{0, 1\}ⁿ \exists t \in S \Rightarrow f(w) \in T. \]

The idea is that the complexity of S is less than or equal to the complexity of T because any membership question for S (i.e., whether w ∈ S) can be easily translated into an equivalent question for T, (i.e., whether f(w) ∈ T). “Easy-to-compute” can be taken as expressible in first-order logic.⁶ We say that two problems have equivalent complexity (S ≡ T) if they are inter-reducible, i.e., S ≤ T and T ≤ S.

2.1 Query resilience
In this paper, we focus on the problem of resilience, a variant of the problem of deletion propagation focusing on Boolean queries: Given D |= q, what is the minimum number of endogenous tuples k to remove from D to make the query false? A larger k implies that the query is more “resilient” and requires the deletion of more tuples to change the query output. In order to study the complexity of resilience, we focus on the decision problem:

Definition 2 (Resilience Decision). Given a query q and database D, we say that (D, k) ∈ Res(q) if and only if D |= q and there exists some set of tuples Γ ⊆ Dⁿ s.t. D − Γ |= q and |Γ| ≤ k.

In other words, (D, k) ∈ Res(q) means that there is a set of k or fewer endogenous tuples whose removal makes the query false. We refer to such a set of tuples Γ as a “contingency set.” Observe that Res(q) ∈ NP since q is computable in PTIME.

A central result of the prior work on resilience [14] is that the complexity of resilience of an sj-free CQ can be exactly characterized via a natural property of its dual hypergraph H(q). The hypergraph of an sj-free query q is usually defined with its vertices being the variables of q and the hyperedges being the atoms [1]. The dual hypergraph, H(q), has vertex set V = \{ g₁, . . . , gₘ \}, and

³Notice that other work sometimes uses dom(D) as the size of the database. Our different definition has no implication on our complexity results but simplifies the discussions of our reductions.

³WLOG, we assume that xᵢ is a tuple of only variables and don’t write the constants. Selections can always be directly pushed into the database before executing the query. In other words, for any constant in the query, we can first apply a selection on each relation and then consider the modified query with a column removed.

³Note that our notion of witness slightly differs from the one commonly seen in provenance literature where a “witness” refers to a subset of the input database records that is sufficient to ensure that a given output tuple appears in the result of a query [8].

⁶In other words, tuples in these atoms provide context and are outside the scope of possible “interventions” in the spirit of causality [17].

³All reductions in this paper are first-order, i.e., when we write S ≤ T we mean S ≡ T. First-order reductions are natural for the relational database setting and they are more restrictive than logspace reductions, which in turn are more restrictive than polynomial-time reductions (S ≡ T ⇒ S ≡ T ⇒ S ≡ T) [21].
each variable $x_i \in \text{var}(q)$ determines the hyperedge consisting of all those atoms in which $x_i$ occurs: $e_i = \{g_j | x_i \in \text{var}(g_j)\}$. A path in the graph is an alternating sequence of vertices and edges, $g_1,x_1,g_2,x_2,\ldots,g_{n-1},x_{n-1},g_n$, such that for all $i$, $x_i \in \text{var}(g_i) \cap \text{var}(g_{i+1})$, i.e., the hyperedge $x_i$ joins vertices $g_i$ and $g_{i+1}$. We explicitly list the hyperedges in the path, because more than one hyperedge may join the same pair of vertices.\footnote{Since we only consider dual hypergraphs, we use the shorter term “hypergraph” from now on.}

Our work on sj-free CQs \cite{14} identified a key structural element of the hypergraph $H(q)$, called a triad, which can fully characterize the complexity of this query class.

Theorem 3 (Dichotomy of resilience for sj-free CQs). Let $q$ be an sj-free CQ and let $q'$ be the result of making all “dominated” atoms exogenous. If $q'$ has a triad, then RES($q$) is NP-complete, otherwise it is in PTIME.

In the remainder of this section, we summarize the intuition behind these main constructs—triads, domination, and linear queries—that lead to the above Dichotomy Theorem. Then, in Section 3 we provide an exposition of how self-join alter or completely invalidate these prior constructs.

2.2 Triads and hardness

We showed in \cite{14} that RES($q_\Delta$) and RES($q_T$) from Fig. 10 are NP-complete. While $q_\Delta$ and $q_T$ appear to be quite different, they share a key common structural property which alone is responsible for hardness for sj-free CQs.

Definition 4 (Triad). A triad is a set of three endogenous atoms, $T = \{S_0, S_1, S_2\}$ such that for every pair $i,j$, there is a path from $S_i$ to $S_j$ that uses no variable occurring in the other atom of $T$.

Intuitively, a triad is a triple of points with “robust connectivity.” Observe that atoms $R, S, T$ form a triad in $q_\Delta$ and atoms $A, B, C$ form a triad in $q_T$ (see Fig. 10). For example, there is a path from $R$ to $S$ in $q_\Delta$ (across hyperedge $y$) that uses only variables (here $y$) that are not contained in the other atom ($y \notin \text{var}(T)$). We showed that triads are responsible for hardness:

Lemma 5 (Triads make RES($q$) hard). Let $q$ be an sj-free CQ where all “dominated” atoms are exogenous. If $q$ has a triad, then RES($q$) is NP-complete.

2.3 Domination

A triad is composed of endogenous atoms only. Some atoms such as $W$ in $q_T$ are given as endogenous, but are never contained in minimal contingency sets. In order to identify triads, we have to simplify the queries by making all such atoms exogenous.

Definition 6 (SJ-free domination). If a query $q$ has endogenous atoms $A, B$ such that $\text{var}(A) \subseteq \text{var}(B)$, we say that $A$ dominates $B$.

Intuitively, whenever a contingency set contains tuples from $B$, we can always replace those with a smaller than or equal number of tuples from $A$.

Proposition 7 (Domination for resilience). Let $q$ be an sj-free CQ and $q'$ the query resulting from labeling some dominated atoms as exogenous. Then RES($q$) $\leq$ RES($q'$).

3 SELF-JOINS CHANGE EVERYTHING

Queries with self-joins are far more complicated than sj-free queries for at least 4 reasons: (1) For the sj-free case, triads alone were shown to determine hardness. Triads need at least 3 existential variables and at least 3 subgoals. Section 3.1 shows that already 2 atoms or 2 variables can be enough for hardness; (2) Linear sj-free queries can be solved using a natural reduction to network flow. For self-join queries, linear queries can be hard. Furthermore, Section 3.3 shows that we may need more elaborate reductions to network flow, even when they are easy. (3) The previous definition of domination does not work in the presence of self-joins. Section 3.2 shows that after replacing the atoms $S, T$ for $q_{rats}$ with $R$, domination does not apply anymore. (4) Our previous crucial concept of the dual hypergraph is no longer sufficient to characterize queries when relations appear multiple times. The position at which a variable appears in a subgoal may influence the complexity of resilience, including whether an atom has repeated variables, e.g., “$R(x, y, R(y, y)$.”

We use a different way of representing queries, which we call binary graphs. Because we focus only on binary queries with self-joins, this representation captures all relevant structural information of the queries, especially the relative position of variables, which the hypergraph representation does not reflect.

Definition 8 (Binary graph). Let $q : A_1, \ldots, A_m$ be a binary CQ. Its binary graph has vertex set $V = \text{var}(q)$ and labeled edge sets defined by atoms $A_1, \ldots, A_m$, i.e. atom $A(x, y)$ translates into edge $x \rightarrow y$. For unary atoms, the edge will be a loop.

Figure 1 illustrates the differences between the dual hypergraph and the binary graph of a binary CQ.
3.1 Basic hard queries: $q_{vc}$ and $q_{\text{chain}}$

We start by proving hardness for two queries that will play an important role in our later results. The first $q_{vc}$ (for "vertex cover") has only two variables and 3 atoms. The second $q_{\text{chain}}$ (since it "chains" two binary relations together) has only 2 atoms and 3 variables:

$$
\begin{align*}
q_{vc} &:= A(x), R(x, y), A(y) & \text{(Vertex cover)} \\
q_{\text{chain}} &:= R(x, y), R(y, z) & \text{(Chain query)}
\end{align*}
$$

Recall that in the sj-free case, a query needs a triad to be hard and all linear queries are easy. In particular, an sj-free query must have at least 3 variables and 3 atoms to be hard.

**Proposition 9 ($q_{vc}$).** $\text{RES}(q_{vc})$ is NP-complete.

**Proof.** A database with unary $A$ and binary $R$ is simply a directed graph, where $A$-tuples are the vertices and $R$-tuples are directed edges. Furthermore, $D \models q_{vc}$ iff the graph $D$ has at least one edge between two vertices. Removing an edge $R(a, b)$ is never better than removing one of the vertices $A(a)$ or $A(b)$. In other words, this problem is unchanged if we make $R$ exogenous, i.e., $\text{RES}(q_{vc}) \equiv \text{RES}(q_{\text{chain}})$ where $q_{\text{chain}} := A(x), R(x, y), A(y)$.

But $\text{RES}(q_{vc})$ is exactly the NP-complete directed vertex cover problem. Therefore, $\text{RES}(q_{vc})$ is NP-complete. □

**Proposition 10 ($q_{\text{chain}}$).** $\text{RES}(q_{\text{chain}})$ is NP-complete.

**Proof.** We reduce 3SAT to $\text{RES}(q_{\text{chain}})$. Let $\psi$ be a 3CNF formula with $n$ variables $x, y, z, \ldots, v_n$ and $m$ clauses $C_1, \ldots, C_m$. We map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $q_{\text{chain}}$: $k_\psi = (2m + 5)n$ and

$$
\psi \in 3\text{SAT} \iff (D_\psi, k_\psi) \in \text{RES}(q_{\text{chain}}).
$$

Figure 11 in Appendix A shows part of $D_\psi$ consisting of the gadgets for $x, y, z, C_1$ where in this example, $C_1 = (x \lor \overline{y} \lor z)$. The variable gadgets are $2m$ cycles whose minimum contingency sets are the set of $m$ blue nodes indicating the variable is assigned true, or the set of $m$ red nodes, indicating the variable is assigned false. The 9-node clause gadgets have minimum contingency sets of size 5 when the clause is assigned true, and 6 otherwise. □

3.2 SJ-Free domination no longer works

We saw from Proposition 7 that in sj-free CQs, dominated atoms (defined as structural criterion in Definition 6) may as well be treated as exogenous. In the presence of self-joins, this criterion no longer works.

**Example 11.** Query $q_{\text{rats}}^{\text{def}} := A(x), R(x, y), R(y, z), R(z, x)$ is a self-join variation of $q_{\text{rats}}$ with $S, T$ replaced by $R$. We have $\text{var}(A) \subseteq \text{var}(R)$, so $A$ dominates $R$, and $R$ should thus become exogenous when

3.3 Easy queries that use flow in a trickier way

As mentioned in the discussion of Theorem 3, resilience for linear sj-free CQ can be computed directly from network flow. As we have just seen, in the presence of self-joins, some linear queries are hard. For those that are easy, network flow can still help us compute resilience, but the arguments become trickier.

To give a sense of this, we present two such examples, the second of which we prove now, the first one in the appendix:

$$
\begin{align*}
q_{\text{conf}}^{AC} &:= A(x), R(x, y), C(z) \\
q_{\text{perm-R}}^{A} &:= A(x), R(x, y), C(z)
\end{align*}
$$

**Proposition 12 ($q_{\text{conf}}^{AC}$).** $\text{RES}(q_{\text{conf}}^{AC})$ is in P.

**Proposition 13 ($q_{\text{perm-R}}^{A}$).** $\text{RES}(q_{\text{perm-R}}^{A})$ is in P.

**Proof.** For a linear sj-free query, we can represent its resilience problem as a network flow making each endogenous tuple an edge of weight 1. Each flow is a join and the min-cuts are exactly the minimum contingency sets (see [32] for details). It is not clear what to do with repeated relations because there is no obvious way to add to a standard network flow algorithm an extra constraint that two
or more edges represent the same tuple, and can thus be removed together at the reduced cost of only 1.

To handle \( q^{\text{perm}}_{A} \), consider an input database of \( A \) and \( R \) tuples.

We refer to \( R \)-tuples that have an inverse as 2-way tuples, and the ones that don’t as 1-way tuples. We construct a flow graph by creating 1-weight edges for all tuples \( A(x) \), and edges \( \{a, b\} \) for pairs of 2-way tuples. There are \( \infty \)-weight edges between an \( A(x) \) and \( \{u, v\} \) if and only if \( x \in \{u, v\} \) or there is a 1-way tuple \( R(x, u) \) or \( R(x, v) \). Note that 1-way tuples are exogenous, since we can always pick an \( A \)-tuple instead, so they have infinite weight in the flow graph.

We show that from the min-cut, \( M \), of the flow graph, we can construct a minimum contingency set, \( \Gamma \), as follows: \( \Gamma \) contains all the \( A(a) \)'s from \( M \). For each edge \( \{a, b\} \in M \), we add one of \( R(a, b) \) or \( R(b, a) \) to \( \Gamma \) as follows: If \( A(a) \in (D - M) \) but \( A(b) \notin (D - M) \) then we add \( R(a, b) \) to \( \Gamma \). Symmetrically, if \( A(b) \in (D - M) \) but \( A(a) \notin (D - M) \) then we add \( R(b, a) \) to \( \Gamma \); otherwise, arbitrarily add one or the other.

We claim that the resulting \( \Gamma \) is a minimum contingency set. Because it comes from a min-cut, it suffices to show that \( \Gamma \) is a contingency set, i.e., \( D - \Gamma \notin q^{\text{perm}}_{A} \). Suppose for the sake of a contradiction, that \( D - \Gamma \) has a join \( A(a), R(a, b), R(b, a) \), i.e., some tuple, \( R(a, b) \), occurs twice in the join. This is impossible because since \( A(a) \notin M \), at least one of \( R(a, b) \) or \( R(b, a) \) must be in \( \Gamma \).

The other possible join is \( A(c), R(c, a), R(a, b) \), \( R(b, a) \). Note that if \( R(c, a) \) is a 1-way tuple, then this would be a flow contradicting the fact that \( M \) is a cut. Thus, \( R(c, a) \) is a 2-way tuple. Since \( A(c) \{c, a\} \) can’t be a flow, the pair \( \{c, a\} \) must be in \( M \).

Since \( R(c, a) \) was not chosen in \( \Gamma \), it must be that \( A(a) \in (D - M) \), which means that there is still a flow from \( A(a) \) to \( \{a, b\} \), so \( M \) was not a cut.

\[ \square \]

\section{4 NEW GENERAL OBSERVATIONS AND PLAN OF ATTACK}

We next give 3 new general observations before we describe our plan of attack in the remainder of the paper.

\subsection{4.1 Minimal queries}

Given queries \( q_1 \) and \( q_2 \), we say that \( q_1 \) is contained in \( q_2 \) (\( q_1 \subseteq q_2 \)) if answers to \( q_1 \) over any database instance \( D \) are always a subset of the answers to \( q_2 \) over \( D \). We say \( q_1 \) is equivalent to \( q_2 \) (\( q_1 \equiv q_2 \)) if \( q_1 \subseteq q_2 \) and \( q_2 \subseteq q_1 \) \([1]\). We say a conjunctive query \( q \) is minimal if for every other conjunctive query \( q' \) such that \( q \equiv q' \) \( q' \) has at least as many atoms as \( q \). For every query \( q \), there exists a minimal equivalent CQ \( q' \) that can be obtained from \( q \) by removing zero or more atoms \([6]\).

From now on, we focus only on minimal queries. This is WLOG, since any non-minimal query can always be minimized as a pre-processing step. The reason is that our hardness evaluation relies on identifying certain subqueries (or patterns) in a query that make this query hard. However, if a pattern is in a subquery that is removed during minimization, then, this pattern has no effect on the resilience of the query.

\subsection{4.2 Query components}

A connected component of \( q \) (or “component” in short) is a non-empty subset of atoms that are connected via existential variables. A query \( q \) is disconnected if its atoms can be partitioned into two or more components that do not share any existential variables. For example,

\[ q^{\text{comp}}_{1} := A(x), R(x, y), R(z, w), B(w) \]

is disconnected and has two components:

\[ q^{\text{comp}}_{1} := A(x), R(x, y) \]

\[ q^{\text{comp}}_{2} := R(z, w), B(w) \]

The resilience of a query is determined by taking the minimum of the resonances of each of its components. In the following, let \( \rho(q, D) \) stand for the resilience of query \( q \) over database \( D \), which is the size of the minimum contingency set for \( q \).

\textbf{Lemma 14 (Query components).} Let \( q := q_1, \ldots, q_k \) be a query that consists of \( k \) components, \( q_i \in \{q\} \). Then \( \rho(q, D) = \min_i \rho(q_i, D) \).

We can now show that the complexity of a query is determined by the hardes of its components if the query is minimal:

\textbf{Lemma 15 (Query components complexity).} Let \( q \) be a minimal query that consists of \( k \) query components. \( \text{RES}(q) \) is \( \text{NP} \)-hard if there is at least one component \( i \in [k] \) for which \( \text{RES}(q_i) \) is \( \text{NP} \)-hard. Otherwise it is in \( \text{P} \).

In the remainder of the paper we assume queries to be connected.

\subsection{4.3 SJ-dominance}

As discussed in Section 3.2, we need to consider the position of the variables in the attribute list of each atom in a sj-query. We write \( \text{pos}_A(i) = x \) to express that the \( i \)-th attribute of atom \( q \) is variable \( x \) for a query \( q \) and omit \( q \) when \( q \) is clear from the context.

\textbf{Definition 16 (Domination).} Let relations \( A \) and \( B \) be endogenous relations in query \( q \). We say that \( A \) dominates \( B \) if there exists a function

\[ f : [\text{arity}(A)] \rightarrow [\text{arity}(B)] \]

such that for each occurrence of \( B \), there exists an occurrence of \( A \) satisfying \( \text{pos}_A(i) = \text{pos}_B(f(i)) \) for \( \forall i \in [\text{arity}(A)] \).

Notice that when \( B \) appears only once, the definition of domination is equivalent to the sj-free definition: \( \text{var}(A) \subseteq \text{var}(B) \).

\textbf{Example 17.} To illustrate the new self-join domination, consider the following queries:

\[ q_1 := R(x, y), A(y), R(y, z), S(y, z) \]

\[ q_2 := R(x, y), A(y), R(y, z), S(y, z) \]

By following the definition above, \( A \) doesn’t dominate \( R \) in \( q_1 \) but it does in \( q_2 \), whereas \( S \) is dominated in both queries. Notice that in \( q_2 \), a tuple \( R(a, b) \) will always join with tuple \( A(b) \) so we can always choose \( A(b) \) instead to be in the contingency set. The same is not true for \( q_1 \), where a tuple \( R(a, b) \) could join with \( A(a) \) or \( A(b) \).

\textbf{Proposition 18 (Domination).} Let \( q \) be a CQ and \( q' \) the result of labeling some dominated relations exogenous. Then \( \text{RES}(q) \subseteq \text{RES}(q') \).
4.4 Outline of our plan of attack

To understand resilience for binary queries in the presence of a single self-join occurring at most 3 times, we proceed as follows. (1) Section 5 shows in general that triads in conjunctive queries with self-joins still imply hardness (Theorem 21) and furthermore, when triads are absent, the endogenous atoms are linearly connected. We call such queries pseudo linear (Theorem 22). We conjecture that pseudo-linear queries may be transformed to linear queries (Theorem 22). We conjecture that pseudo-linear queries may be transformed to linear queries of equivalent resilience (Conjecture 23). In any case, it suffices to study the criteria for hardness of pseudo-linear queries. (2) Section 6 generalizes the hardness pattern behind $q_{i}$ to a more general class of hard queries that contain "paths" between repeated atoms. (3) We then focus on the complexity of the resilience of binary CQs with at most a single repetition of a single relation. Section 7 gives a complete characterization of the complexity for the cases of 2 occurrences of a repeated atom. This is a dichotomy theorem: we show that for all such queries, $q$, RES($q$) is either NP-complete or RES($q$) is reducible to network flow and is thus in P. Section 8 presents the remaining challenges that must be overcome in order to characterize all queries with the sj-relation occurring 3 times.

5 NON-LINEAR QUERIES: NP-COMPLETE

In this section we prove that queries containing triads remain hard in the presence of self-joins (Theorem 21). We then show that for any query that does not contain a triad, its endogenous atoms are arranged linearly. We call such a query pseudo-linear. Thus, we conclude that either a query contains a triad in which case its resilience problem is NP-complete, or it is pseudo-linear. In the following sections, we can thus safely restrict our attention to pseudo-linear queries.

We first observe that self-joins can only make the resilience of queries harder:

**Lemma 19 (SJ Can Only Make Resilience Harder).** Let $q$ be an sj-free CQ and let $q'$ result from $q$ by adding self-joins, i.e., some atoms $S_{i}(\overline{v})$ from $q$ are replaced by the atom $R_{i}(\overline{v})$ where the relation $R_{i}$ occurs elsewhere in $q$. If $q'$ is minimal, then RES($q'$) ≤ RES($q$).

**Proof.** Let $D \models q$ be a database. We map $D$ to $D'$ by marking all the tuples according to which variables they refer to in joins of $q$. For each join $j$ assigning the variables of $q$ to a witness $D \models q$, we add the tuples $T(j(v_{1}), \ldots , j(v_{k}))$ to $D'$, where $T(\overline{v})$ occurs in $q'$. In particular, $S_{i}(j(v_{1}), \ldots , j(v_{k})) \in D$ results in the tuple $R_{i}(j(v_{1}), \ldots , j(v_{k})) \in D'$.

Since the variables mark the tuples in $D'$, the new self-joins have no effect: if the subscripted variables are $\overline{v}$ in a tuple of $R_{i}$ in $D'$, then it came from a tuple of $S_{i}$ in $D$. It then follows that there is a 1:1 correspondence of contingency sets for $(D, q)$ and $(D', q')$. We need the minimality of $q'$, because if there were an assignment where $D - \Gamma' \models q'$ when $D - \Gamma \not\models q$, this would correspond to a reassignment of the variables, var($q'$) to a proper subset, so that some $R_{i}$ would be doing "double duty". This would mean that a proper subset of $q'$ implies $q'$, i.e., $q'$ is not minimal.

It does not immediately follow from Lemma 19 that every query with a triad is hard. The missing cases are where an sj-query includes a triad, but it doesn’t come from an sj-free query with a triad because in any such sj-free query, some of the triad’s atoms are dominated. We next explore this situation.

5.1 Self-join variations of $q_{\triangle}$, $q_{\mathit{rats}}$ and $q_{\mathit{brats}}$

Recall three important sj-free queries:

- $q_{\triangle} := \Delta(R(x, y), S(y, z), T(z, x))$ (Triangle)
- $q_{\mathit{rats}} := A(x), R(x, y), S(y, z), T(z, x)$ (Rats)
- $q_{\mathit{brats}} := A(x), R(x, y), B(y), S(y, z), T(z, x)$ (Brats)

RES($q_{\triangle}$) is NP-complete because it contains the triad, $R, S, T$. However $q_{\mathit{rats}}$ and $q_{\mathit{brats}}$ are easy because $A$ dominates $R$, $T$ and $B$ dominates $S$ so they only have two endogenous atoms each and thus no triad.

Figure 3 shows two self-joinings of $q_{\triangle}$. By Lemma 19, the resilience of these and any other self-joinings of $q_{\triangle}$ are NP-complete.

Figure 4 shows two self-joinings of $q_{\mathit{rats}}$. The self-join relation $R$ is now more robust. It is not dominated by $A$, so these two queries are hard: they still contain triads consisting of their three $R$’s. Since they have a triad, these queries are hard but we cannot use Lemma 19 to show this because their sj-free counterpart is easy. Below we list some $q_{\mathit{rats}}$ and $q_{\mathit{brats}}$ self-join variations which contain triads, and proceed to show their complexity is hard.

- $q_{\mathit{rats}}^{s_{1}} := A(x), R(x, y), R(y, z), R(z, x)$
- $q_{\mathit{rats}}^{s_{2}} := A(x), R(x, y), R(y, z), R(x, z)$
- $q_{\mathit{brats}}^{s_{1}} := A(x), R(x, y), B(y), R(y, z), R(z, x)$
- $q_{\mathit{brats}}^{s_{2}} := A(x), R(x, y), B(y), R(y, z), R(x, z)$
- $q_{\mathit{brats}}^{s_{3}} := A(x), R(x, y), R(y, z), R(z, x), B(z)$

**Proposition 20.** Let $q$ be a self-join variation of $q_{\mathit{rats}}$ or $q_{\mathit{brats}}$. If $q$ has a triad, then RES($q$) is NP-complete.
Proof. The proofs essentially follow the same strategy used to reduce 3SAT to $\text{RES}(q_{\Delta})$ with a few adjustments to handle the self-joining relation and also the variable order, which is relevant in some cases. See Lemma 46 and Lemma 47 for the details. \(\blacksquare\)

Using Proposition 20, we now generalize the fact that triads make sj-free queries hard (Lemma 5) to the same result for general CQs.

5.2 Triads Make Queries Hard

**Theorem 21 (SJ-queries with triads).** If $q$ has a triad, then $\text{RES}(q)$ is NP-complete.

Proof. This mostly follows from the fact that triads make sj-free queries hard and adding self-joins to a hard query keeps it hard (Lemma 5, Lemma 19).

The case we haven’t covered yet is where the triad in $q$ involves self-join relations which would be dominated and thus exogenous in the corresponding sj-free query. Examples are self-join versions of $q_{\text{rats}}$ and $q_{\text{beats}}$ which are hard even though – because of domination – their sj-free cases are easy (Proposition 20).

We now follow and extend the proof of Lemma 5 when $q$ has a triad, $T = (S_0, S_1, S_2)$, even though if $T$ did not include a self-join, one or more of its members would be dominated. In Case 1, $\text{var}(S_i)$, $i = 1, 2, 3$, are pairwise disjoint. Here the reduction from $\text{RES}(q_{\Delta})$ to $\text{RES}(q)$ goes through exactly as in the proof of Lemma 5. We can choose a single relevant variable for each $S_i$, so no domination is possible. Any minimum contingency set consists of elements of $S_0((ab)), S_1((bc))$ or $S_2((ca))$, and the reduction from $\text{RES}(q_{\Delta})$ goes through.

In Case 2, where $\text{var}(S_i)$ are not pairwise disjoint, we have to consider a partition of the variables into 7 pieces (Eqn. 6 from the proof of Lemma 5). As argued there, there is still a 1:1 correspondence between joins of $(D, q_{\Delta})$ and joins of $(D', q')$.

If there are no (endogenous) relations containing just the $a$, $b$ or $c$ variables, then the reduction from $\text{RES}(q_{\Delta})$ goes through. If there is a relation containing just $a$, then we instead use the same reduction but from the appropriate self-join variation of $q_{\text{rats}}$. If there are relations containing just $a$ and $b$ but not $c$, then we get a reduction from the appropriate self-join variation of $q_{\text{beats}}$. If there are relations for $a$, $b$ and $c$, then these form an sj-free triad and thus we already know that $\text{RES}(q)$ is hard. \(\blacksquare\)

Thus, if a query contains a triad it is hard. Otherwise, we only need to focus on its endogenous atoms, which are linearly connected.

5.3 No Triad Means Pseudo-Linear

In [14], we proved that if a sj-free CQ, $q$, has no triad, then $q$ may be transformed to a sj-free CQ query $q'$ which is linear and such that $\text{RES}(q') \geq \text{RES}(q)$. Since linear sj-free CQ’s are easy, it follows that $q$ is easy.

This argument no longer works in the presence of self-joins because linear queries can be easy or hard. However, we can extend the theorem from [14] to show the following.

**Theorem 22 (No Triad Means Pseudo-Linear).** Let $q$ be a CQ with no triad. Then all endogenous atoms in $q$ are connected linearly.

![Figure 5: A walk along the endogenous atoms. The cut $c_i$ results from removing all the variables (edges) from group $G_i$.](image)

Proof. We are given $q$, a CQ with no triad. Let $n$ be the number of groups of endogenous atoms in $q$, where we put two atoms in the same group iff they contain exactly the same variables, so $A(x, y)$ and $R(y, x)$ belong in the same group, but $A(x, z)$ and $R(x, y)$ do not.

Since $q$ is connected but has no triad, the groups of endogenous atoms, $G_1, G_2, \ldots, G_n$ may be ordered so that for all $i$ with $2 \leq i \leq n - 1$, removing the variables of $G_i$ separates the atoms of $q$ into two connected components, with $G_1, \ldots, G_{i-1}$ in one and $G_{i+1}, \ldots, G_n$ in the other, see Figure 5.

Note, that if $A$, $B$, $C$ are endogenous atoms from different groups and $A$ and $B$ are already placed along the line, say with $B$ to the right of $A$ then it is easy to see where $C$ must go. Since $q$ is connected and $A$, $B$, $C$ is not a triad, the variables of exactly one of $A$, $B$, $C$ separate the other two. If $A$ is the separator, $C$ goes to the left of $A$, if $B$ is the separator, $C$ goes to the right of $B$ and if $C$ is the separator, then it goes between $A$ and $B$, and that’s what guarantees the exogenous atoms are linearly connected.

Looking at Figure 5, we see that the the endogenous atoms of $q$ are arranged linearly. The jobs of the exogenous atoms of $q$ are at most to connect the endogenous atoms, and also, if necessary, ensuring that $q$ is a minimal query. \(\blacksquare\)

We believe a stronger claim holds:

**Conjecture 23 (No Triad Means Linear).** Let $q$ be a CQ with no triad. Then we can transform $q$ to a linear CQ $q'$ with $\text{RES}(q) \equiv \text{RES}(q')$.

6 PATHS ARE HARD

Section 3.1 presented two linear queries that are hard, unlike in the sj-free case where all linear queries are easy. We now identity a pattern characteristic of $q_\text{vc}$ that we call a path. The main result of this section is that every query containing a path is hard.

Recall that we restrict our attention to binary queries with at most one repeated relation or a single-self-join (ssj). We first show that if the self-join relation is unary, then the query is always hard!

**Theorem 24 (Unary Path).** Let $q$ be a minimal ssj-CQ. If $q$ contains distinct atoms $A(x)$ and $A(y)$, then $\text{RES}(q)$ is NP-complete.

Proof sketch. Let $A(x)$ and $A(y)$ be the first two occurrences of the relation $A$ in $q$. Since $q$ is connected, $A(x)$ and $A(y)$ are connected by at least one non-self-join relation, $R$ (see Fig. 6a). We prove that $\text{RES}(q_\text{vc}) \leq \text{RES}(q)$. Details are in Appendix A, but it is not hard to see that any database $D \models q_\text{vc}$ can be transformed to a database $D' \models q$ that exactly preserves resilience. Here $A', R'$ in $D'$ come from $A$ and $R$ in $D$, and all the other atoms of $q$ (including any
additional occurrences of the self-join relation, $A$, to the right of $A(y)$ are covered by multiple, extra values which complete the joins but are never chosen in minimum contingency sets. Note that this proof doesn’t make any assumption about the existence or not of triads.

When the self-join relation is binary, if two consecutive atoms, $A(x, y), A(z, w)$, are disjoint, then we call this a binary path. "Overlapping" consecutive atoms with shared variables, such as $R(x, y), R(y, z)$ in $q_{chain}$, can also cause hardness and are studied in later sections.

**Theorem 25 (Binary Path).** Let $q$ be a minimal ssj-CQ. If $q$ has distinct consecutive sj atoms $A(x, y), A(z, w)$ with $\{x, y\} \cap \{z, w\} = \emptyset$, then $RES(q)$ is NP-complete.

**Proof Sketch.** Given $A(x, y), A(z, w)$ as in the statement of the theorem, there must be an atom $R(u, v)$, with $R \neq A$ on the path between them, and $u \in \{x, y\}$ and $v \notin \{x, y\}$. Now, as in the proof of Theorem 24, we reduce $RES(q_{wc})$ to $RES(q)$. We map any database $D \models q_{wc}$ to a database $D' \models q$, where $A'$ contains $\{(a, a) \mid A(a) \in D\}$ plus other multiple, extra values for any other atoms of the relation $A$ in $q$ to the left of $A(x, y)$ or the right of $A(z, w)$ and $R' = \{(a, b) \mid R(a, b) \in D\}$. Same as in the unary case, there is no assumption about the linearity of the query. Details are in Appendix A.

Unary and Binary Paths are the simplest of the hard patterns. By Theorem 24 and Theorem 25, they always force their queries to be hard. In the next sections we study the more subtle linear ssj-queries, which do not contain paths.

### 7 QUERIES WITH EXACTLY TWO $R$-ATOMS

In this section we cover the complexity of binary linear ssj-queries with exactly two self-joining atoms. As always, we assume that our query is minimal and connected, and from now on also assume that $q$ does not contain a triad or a path as described in Theorem 24 and Theorem 25; otherwise we would already know that $RES(q)$ is NP-complete. Even in this restricted setting, we will see that there is a surprisingly rich variety of structures, requiring different strategies to determine their complexity.

Because there are no paths, the two sj-atoms must be binary and must have at least one variable in common.

- **Chains** have one common variable and join in different attributes, e.g., $R(x, y), R(y, z)$;
- **Confluences** have one common variable and join in the same attribute, e.g., $R(x, y), R(y, z)$;
- **Permutations** share both variables but join in different attributes, e.g., $R(x, y), R(y, x)$;
- Queries with repeated variables (REP) have repeated variables in at least one sj-atom, e.g., $R(x, x), R(x, y), B(y)$

We consider each of these possibilities in turn and characterize their complexity.

#### 7.1 2-Chains

The chain query is the simplest possible minimal sj-query with two atoms and we proved earlier that its resilience is NP-complete (Proposition 10). In this section we prove that the chain structure is quite robust and that any of its variations remains NP-complete.

We call "variations" of $q_{chain}$ any query obtained by adding new relations to it, i.e. relations that do not self-join. We start by presenting the variations obtained by adding unary relations and then generalize that to any variation.

Figure 7a shows how unary relations can be added to $q_{chain}$. Each one can appear by itself or combined with others. While the proof involves several subcases, the important take-away is that all 8 of these variations are hard.

**Proposition 26 (Chains with unary relations).** Any variation of $q_{chain}$ with unary relations is NP-complete.

**Proof.** We prove these variations are hard by a reduction from 3SAT. The same idea used to prove that $RES(q_{chain})$ is hard will work here as long as we adapt the variable and clause gadgets to deal with the existence of the unary relations. Lemmas 48 to 50 in Appendix A contain the details.

Now we can generalize this hardness result to any chain variation using a reduction idea similar to the ones used for the proofs of Theorems 24 and 25 for paths.

**Proposition 27 (Chains).** If a query $q$ contains a 2-chain as its only self-join, then $RES(q)$ is NP-complete.

#### 7.2 2-Confluences

Confluences are defined by a relation self-joining only in the same attribute. We refer to this pattern as $q_{conf}$ (Fig. 7b).

Note that as a stand-alone query $q_{conf}$ is not minimal, so we need other atoms connected to both $x$ and $z$. An example of a minimal query containing a confluence is $q_{conf}^AC \equiv A(x), R(x, y), R(z, y), C(z)$.

We next show that the standard fow algorithm without any modifications works correctly for linear queries with no self-join other than one 2-confluence, thus generalizing the idea of Proposition 12.

**Proposition 28 ($q_{conf}$).** $RES(q)$ for any linear query $q$ with $q_{conf}$ as its only self-join can be solved in $PTIME$ by standard network flow.

**Proof.** For $q \models q_{conf}$, let $D$ be any database satisfying $q$ and let $j$ be a join of $D$ satisfying $q$. Note that if $y$ occurs in $q_j$ then, by linearity, it must be as an atom $f(x, y)$ immediately to the left of $R(x, y)$. Furthermore, any such atom may be considered...
enough because it is never better to choose \( F(a, b) \) over \( R(a, b) \). Furthermore, if \( x \) occurs in \( q_m \), then it would be via an atom \( F(x, y) \) immediately to the right of \( R(x, y) \). If so, we can assume it is immediately to the left of \( R(x, y) \). In particular, we may assume that neither \( x \) nor \( z \) occurs in \( q_m \).

We can write \( j = A(a, b)R(a, b)B(b)R(c, b)C(c, b) \) where \( A(a, b) \), \( B(b) \), \( C(c, b) \) stand for the atoms of \( q_T(a, b), q_m(b), q_r(b, c) \), respectively.

Let \( N_D \) be a network flow for \( D, q \) ignoring the fact that \( q \) has a self-join. Thus \( N_D \) duplicates edges for its \( R \)-tuples, i.e., for each \( R(a, b) \in D \) there are two edges, \( R_x(a, b), R_y(a, b) \) in \( N_D \). Assume that each edge corresponding to an endogenous, resp. exogenous tuple has weight 1, resp. \( \infty \).

Let \( M \) be a min cut for \( N_D \). Let \( \Gamma_M \) be the corresponding set of atoms of \( D \). We claim that in fact \( \Gamma_M \) is a minimum contingency set for \( (D, q) \). The key idea is the following:

**Lemma 29.** Let \( M \) be a minimal cut of \( N_D \). Then \( M \) does not include more than one instance of any \( R \)-tuple.

**Proof.** Suppose to the contrary, that \( M \) is a minimal cut for \( N_D \) and contains both \( R_x(a, b) \) and \( R_y(a, b) \). Since \( M \) is minimal, it follows that \( N_D - \{M - \{R_x(a, b)\}\} \) and \( N_D - \{M - \{R_y(a, b)\}\} \) both contain flows:

\[
\begin{align*}
&f_1 = A(a, b)R_x(a, b)B(b)R_y(c, b)C(c, b) \quad \text{and} \\
&f_2 = A(a', b)R_y(a', b)B(b)R_x(c, b)C(c, b).
\end{align*}
\]

But then \( N_D - M \) contains the flow \( f = A(a', b)R_y(a', b)B(b)R_x(c, b)C(c, b) \), contradicting the fact that \( M \) is a cut. \( \square \)

Now, let \( \Gamma \) be any contingency set. We claim that \( \Gamma \) is the same size as some cut of \( N_D \). To see this, let us first let \( S \) be the result of replacing each atom \( R(a, b) \in \Gamma \) with both possible edges, \( R_x(a, b), R_y(a, b) \) in \( N_D \). Since \( \Gamma \) is a contingency set, it follows that \( S \) is a cut of \( N_D \). Now, let \( S' \) be a minimal subset of \( S \) that is still a cut, where some of the extra \( R \)-edges, i.e., either \( R_x(a, b) \) or \( R_y(a, b) \) have been removed.

By the proof of Lemma 29, we know that \( S' \) has only one edge for each atom \( R(a, b) \in \Gamma \). Thus, \( |S'| = |\Gamma| \) as claimed. It follows that the size of a min cut of \( N_D \) is the same as the size of a minimum contingency set for \( (D, q) \). \( \square \)

7.3 2-Permutations

We call two \( R \)-atoms sharing both variables a permutation. The smallest pattern that has this property is \( R(x, y), R(y, x) \). We show that permutations have both NP-complete and PTIME instances.

### Easy permutations

We start with two easy permutations (Fig. 8a).

\[ q_{perm} := R(x, y), R(y, x) \]

\[ q_{perm}^A := A(x), R(y, x), R(y, x) \]

**Proposition 30.** \( \text{RES}(q_{perm}) \) and \( \text{RES}(q_{perm}^A) \) are in \( P \).

**Proof.** Given a database \( D_1 \) satisfying \( q_{perm} \), each tuple that is part of a witness for \( D_1 \), \( q_{perm} \) is part of exactly one witness. Therefore the size of a minimum contingency set for \( D_1 \), \( q_{perm} \) is exactly the number of witnesses.

Given a database \( D_2 \) satisfying \( q_{perm}^A \), we have two possible choices. Either \( A(a) \) will be in the min \( \Gamma' \) or either one of \( R(a, b) \) and \( R(b, a) \) but not both. Therefore we can reduce \( \text{RES}(q_{perm}^A) \) to vertex cover in a bipartite graph, which in \( P \). \( \square \)

### Hard permutations

Surprisingly, adding another unary atom to \( q_{perm}^A \) thus bounding it on both ends (Fig. 8b), leads to a hard query.

\[ q_{perm}^{AB} := A(x), R(x, y), R(y, x), B(y) \]

It is still true that for any pair \( R(a, b), R(b, a) \) participating in a join, a minimum contingency set will only contain one tuple from the pair. This might lead to the wrong conclusion that network flow could solve this problem. We will next show that this is incorrect.

**Proposition 31.** \( \text{RES}(q_{perm}^{AB}) \) is NP-complete.

**Proof Sketch.** As in Proposition 10, we define a reduction from 3SAT by creating variable and clause gadgets. An important feature of how tuples join in this case is the fact that a tuple \( R(a, b) \) is never in a minimum contingency set unless \( A(a), A(b), B(a), B(b) \) are all present. The proof is in Appendix A. \( \square \)

**The criterion.** The main structural difference between the hard and easy permutations defined above is whether or not there are relations that “bound” the permutation on both ends, i.e. whether there are endogenous relations \( S, T \), such that \( S \) contains variable \( x \) but not \( y \), and \( T \) contains variable \( y \) but not \( x \). Thus, the hard permutation, \( q_{perm}^{AB} \), is bound, but the easy ones, \( q_{perm}, q_{perm}^A \), are not bound. Using this characterization, we identify when 2-permutations are hard.

**Proposition 32.** Let \( q \) be a query with \( R(x, y), R(y, x) \) as its only self-join. If \( q \) is bound, then \( \text{RES}(q) \) is NP-complete; otherwise, \( \text{RES}(q) \) is in \( P \).

7.4 Queries with REP

We call queries with repeated variables (or REP in short) those where sj-atoms contain the same variable twice, e.g. occurrences of \( R(x, x) \).

There are only 3 such cases, depending on whether one or both sj-atoms have REP’s, and whether or not the sj-atoms have disjoint sets of variables:

\[ z_1 := R(x, y)S(x, y)R(y, y) \]
\[ z_2 := R(x, y)S(x, y)R(y, z) \]
\[ z_3 := R(x, y)R(y, x)A(y) \]
Notice that queries $q_1$ and $q_2$ have a binary path and are thus NP-complete. We show next that $\text{RES}(q_3)$ is in \( P \).

**Proposition 33.** Any linear query with exactly 2 sj-atoms that contains $q_3$ is in \( P \).

**Proof.** Given a database $D$ such that $D \models q_3$, joins can occur in two ways:

\[(a, a, a) = (R(a, a), A(a))\]
\[(a, a, b) = (R(a, a), R(a, b), A(b))\]

From that, we can conclude that no tuple $R(a, b)$ with $a \neq b$ needs to be in a contingene set, since we can choose either $R(a, a)$ or $A(b)$ instead. Thus, since the $R(a, b)$ is essentially exogenous, $q_3$ and any linear extension of it can be solved with network flow. \( \square \)

### 7.5 The dichotomy

Thus, combining our results so far, with at most two occurrences of the self-join relation, we have nearly proved a complete characterization of the complexity of resilience:

**Theorem 34 (Two-Atom Dichotomy).** For $q$ an ssj-CQ, with at most two occurrences of the self-join relation, $\text{RES}(q)$ is either NP-complete or reducible to network flow and thus inside \( \text{PTIME} \). In particular there is a \( \text{PTIME} \) algorithm that on input $q$ determines which case occurs.

In all but one of our results above, pseudo-linearity suffices. The only factor is in Proposition 28 where we used the assumption that $q$ was linear, thus guaranteeing that every path in $q$ from $x$ to $z$ involves the variable $y$. Note that this is not true for the query $q_{cf} := \exists x \in R(y) \land \exists x \in R(z, y)$. It is easy to see that $q_{cf}$ is pseudo-linear and $\text{RES}(q_{cf}) \equiv \text{RES}(q_{vc})$. Thus, we complete the proof of Theorem 34 by observing,

**Proposition 35.** Let $q$ be a pseudo-linear query $q$ with $q_{conf}$ as its only self-join. If $q$ contains an exogenous path from $x$ to $z$ not involving the variable $y$, then $\text{RES}(q)$ is NP-complete; otherwise it is in \( \text{PTIME} \).

### 8 Queries With Exactly Three $R$-Atoms

In Theorem 34 we completely characterized the complexity of resilience of all CQs with at most one repetition of a single relation, thus extending the dichotomy for sj-free CQs into the land of self-joins.

In this section, we present an overview of what can happen when we allow a third $R$-atom to self-join. Since we only have to consider pseudo-linear queries that do not have a path, all three $R$-atoms must connect to each other directly or through the third $R$-atom. Even though this is still a restrictive setting, we will see that it brings non-trivial complications to the characterization. We will present some complexity results; but also some remaining open problems.

#### 8.1 3-Chains

We obtain a 3-chain by adding an extra $R$-atom to a 2-chain in a way such that the new atom joins in a different attribute from the other two.

\[q_{\text{chain}} := R(x, y), R(y, z), R(z, w)\]

Analogous to the 2-chain case, 3-chains are always hard. In fact this holds for 4-chains, 5-chains, etc.

**Proposition 36.** For all $k \geq 2$, if $q$ contains a $k$-chain as its only self-join, then $\text{RES}(q)$ is NP-complete.

### 8.2 3-Confluences

Adding a third $R$-atom to a 2-confluence and making sure that it joins in the same attribute with one of the two existing $R$-atoms produces a 3-confluence.

\[q_{\text{conf}} := R(x, y), R(z, y), R(z, w)\]

As in the 2-confluence case, $q_{\text{conf}}$ is not minimal, so other atoms are required to make it minimal. Here are a few examples of minimal queries containing $q_{\text{conf}}$.

\[q_{\text{AC}} := A(x), R(x, y), R(z, y), R(z, w), C(w)\]
\[q_{\text{AS}} := T(x, y)^S, R(x, y), R(z, y), R(z, w), S(z, w)^S\]

These queries are very similar but one of them is hard, while the other one is easy.

**Proposition 37.** $\text{RES}(q_{\text{AC}})$ is NP-complete.

**Proposition 38.** Any variation of $q_{\text{AC}}$ obtained by including unary relations is NP-complete.

**Proof.** We define a reduction from Max 2SAT similar to the one used for $q_{\text{AC}}$ by adding the appropriate tuples to obtain the same set of joins. The contingency set doesn’t change with the new tuples and therefore the properties of the reduction hold. \( \square \)

**Proposition 39.** $\text{RES}(q_{\text{AS}})$ is in \( P \).

**Open Problem.** There is a third variant of 3-confluences which somewhat mix queries $q_{\text{AC}}$ and $q_{\text{AS}}$ (Fig. 9).

\[q_{\text{SC}} := A(x), R(x, y), R(z, y), R(z, w), S(z, w)^S\]

The complexity of $\text{RES}(q_{\text{SC}})$ remains unknown.

### 8.3 3-Chain-Confluence

With 3 $R$-atoms, it is possible that different patterns will occur at the same time. This feature of this case makes it harder to analyze the queries, since the result of these interactions might diverge from what we expect when we see each pattern in isolation.

In this section we present some queries where a 2-chain and a 2-confluence occur at the same time.

\[q_{\text{AC}} := A(x)R(x, y)R(y, z)R(w, z)C(w)\]
\[q_{\text{AS}} := A(x)R(x, y)R(y, z)R(w, z)S(w, z)\]
\[q_{\text{SC}} := R(x, y)R(y, z)R(w, z)C(w)\]

The resilience of these queries is hard but they require different reductions. If $x$ is bound, then we can use a reduction from $\text{RES}(q_{\text{chain}})$. Otherwise we need a reduction from Max 2SAT.

**Proposition 40.** $\text{RES}(q_{\text{AC}})$ and $\text{RES}(q_{\text{AS}})$ are NP-complete.

**Proposition 41.** $\text{RES}(q_{\text{SC}})$ is NP-complete.

**Open Problem.** In this category of queries with chain and confluence, we don’t know the complexity of $q_{\text{SC}} := R(x, y)R(y, z)R(w, z)S(w, z)$. 

Cilele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou
3-Permutation plus R

It is not possible to obtain two permutations in a query with only 3 R-atom. In fact, there are only two way that a new R-atom can be connected to a permutation: either by joining with x or y, and those are equivalent.

$$q_{\text{perm-R}} := R(x, y), R(y, z), R(z, y)$$

Similar to the $q_{\text{conf}}$ case, $q_{\text{perm-R}}$ is not a minimal query, so additional atoms are necessary. We list the main examples of how this query can be made minimal and discuss the complexity of their resilience.

First we start with a query we have already seen and another one that is a slight variation on the first (Fig. 2b).

$$q_{\text{perm-R}}^A := A(x)R(x, y)R(y, z)R(z, y)$$
$$q_{\text{perm-R}}^{3\text{perm-R}} := S(w, x)R(x, y)R(y, z)R(z, y)$$

We proved in Proposition 43 that RES($q_{\text{perm-R}}^A$) is in P by using network flow. A similar argument proves that RES($q_{\text{perm-R}}^{3\text{perm-R}}$) is also in P.

**Proposition 42.** RES($q_{\text{perm-R}}^{3\text{perm-R}}$) is in P.

The next query we will see is $q_{\text{perm-R}}^{S_{xy}}$. Although very similar to $q_{\text{perm-R}}^A$ and $q_{\text{perm-R}}^{3\text{perm-R}}$, RES($q_{\text{perm-R}}^{S_{xy}}$) is hard. It is surprising that such a small difference can already change the complexity of the resilience problem. Moreover, the proof requires a new reduction instead of a reduction similar to the one used in Proposition 31.

$$q_{\text{perm-R}}^{S_{xy}} := S^S(x, y)R(x, y)R(y, z)R(z, y)$$

**Proposition 43.** RES($q_{\text{perm-R}}^{S_{xy}}$) is NP-complete.

Some other examples of queries that are hard but these are somewhat related to $q_{\text{perm}}$.

$$q_{\text{perm-R}}^{AC} := A(x)R(x, y)R(y, z)R(z, y)C(z)$$
$$q_{\text{perm-R}}^{AB} := A(x)R(x, y)B(y)R(y, z)R(z, y)$$
$$q_{\text{perm-R}}^{BC} := S(x, y)R(x, y)B(y)R(y, z)R(z, y)C(z)$$

**Proposition 44.** RES($q_{\text{perm-R}}^{AC}$), RES($q_{\text{perm-R}}^{AB}$) and RES($q_{\text{perm-R}}^{BC}$) are NP-complete.

**Open Problems.** Despite the similarities with the queries presented in this section, we were not able to determine the complexity of the following queries:

$$q_{\text{perm-R}}^{AS_{xx}} := A(x)S(x, y)R(x, y)R(y, z)R(z, y)$$
$$q_{\text{perm-R}}^{SB_{xy}} := S(x, y)R(x, y)B(y)R(y, z)R(z, y)$$
$$q_{\text{perm-R}}^{BC} := S(x, y)R(x, y)R(z, y)R(z, y)C(z)$$

8.5 Queries with REP

If all three occurrences of R have repeated variables, then we are in the path case.

$$z_4 := R(x, y)R(x, y)S(x, y)R(y, y)$$
$$z_5 := A(x)R(x, y)R(y, z)R(z, z)$$

**Proposition 45.** RES($z_4$) and RES($z_5$) are NP-complete.

**Open problems.** We don’t know the complexity of other queries that fall in this category of having three R-atoms with REP but the following open ones are intriguing.

$$z_6 := A(x)R(x, y)R(y, z)C(z)$$
$$z_7 := A(x)R(x, y)R(y, z)R(y, y)$$

Query $z_6$ has a similar structure to $q_{\text{chain}}$ but a similar reduction doesn’t seem to work. Similarly, a reduction from RES($q_{\text{perm}}^{AB}$) doesn’t work for $z_7$.

9 RELATED WORK

In prior work [14], we identified the concept of a triad, a novel structure that allowed us to fully characterize the complexity of resilience (and consequently for deletion propagation) for the class of self-join-free conjunctive queries with potential functional dependencies. Our work in this paper considers self-joins, which have long-plagued the study of many problems in database theory; results for such queries have been few and far between.

**Deletion propagation and view updates.** The problem of resilience is a special case of deletion propagation, focusing on Boolean queries. Deletion propagation generally refers to non-Boolean queries. Given a non-Boolean query q and database D, the typical goal is to determine the minimum number of tuples that must be removed from D, so that a tuple t is no longer in the query result [5, 12] (source side-effects). Variants of deletion propagation consider side-effects in the query result rather than the source [23, 24], and multi-tuple deletions [9, 25]. Resilience and deletion propagation are special cases of the view update problem [3, 9, 10, 12, 13, 15, 22], which consists of finding the set of operations that should be applied to the database in order to obtain a certain modification in the view.

**Causality and explanations.** Database causality is geared towards providing explanations for query results, but typically relies
on the concept of responsibility [32, 33], which is harder than resilience. The idea of interventions appears in other explanation settings, but often apply to queries instead of the data [34, 35, 37]. Finally, the problem of explaining missing query results [7, 18–20, 36] is a problem analogous to deletion propagation, but in this case, we want to add, rather than remove tuples from the view.

**Provenance and view updates.** Data provenance studies formalisms that can characterize the relation between the input and the output of a given query [4, 8, 11, 16]. "Why-provenance" is the provenance type most closely related to resilience. The motivation behind Why-provenance is to find the "witnesses" for the query answer, i.e., the tuples or group of tuples in the input that can produce the answer. Resilience, searches to find a minimum set of input tuples that can make a query false.

10 **FINAL REMARKS**

In this paper, we studied the problem of resilience for binary conjunctive queries with self-joins. We identified fundamental query structures that impact hardness, and proved a complete dichotomy for the restricted class of binary CQs where at most two atoms can correspond to the same relation.

Our work also creates a roadmap for tackling the analysis of more extended query families. Section 8 provides significant results towards the generalization for the case of binary CQs with a single self-join relation that appears in 3 atoms, and identifies the remaining open problems and challenges towards completing the dichotomy for this class.

We also see the generalization to conjunctive queries without arity restrictions as a natural extension. As a first step towards that direction, one can consider queries where up to two attributes per relation may participate in joins, thus allowing for a reduction to the binary relation case by projecting to the joining columns. To handle the resulting bag semantics, the flow algorithm would need to be modified with appropriate weights. Special attention would need to be given to the network flow algorithm used in Proposition 13 since just an adjustment on weights would not be enough to result in a correct solution.

Overall, our work in this paper contributes important progress in the theoretical analysis of self-joins, which has long been stalled for many related problems. We hope that our results, even though they apply to a restricted class, will provide the foundations to help solve the general case for CQs with self-joins in the future.

---

**References**

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases: The Logical Level*. Addison-Wesley.

[2] Antoine Amarilli, Mikkel Monet, and Pierre Senellart. 2017. Conjunctive Queries on Probabilistic Graphs: Combined Complexity. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS’17)*. ACM, New York, NY, USA, 217–232. https://doi.org/10.1145/3047486.3056121

[3] P. Bancelhon and N. Spirat. 1981. Update Semantics of Relational Views. *ACM TODS* 6, 4 (Dec. 1981), 357–375. https://doi.org/10.1145/1.38628.319834

[4] Peter Buneman, Sanjeev Khanna, and Wang Chiwei Tan. 2001. Why and Where: A Characterization of Data Provenance. In *ICDT ’01*. 316–330.

[5] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. 2002. On Propagation of Deletions and Annotations Through Views. In *PIDS* 150–158. https://doi.org/10.1007/3-540-456363

[6] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC* 77–90. https://doi.org/10.1145/800150.803897

[7] Adrianne Chapman and H. V. Jagadish. 2009. Why not? In *SIGMOD* 523–534.

[8] James Cheney, Lucia Chiticariu, and Wang Chiwei Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases* 1, 4 (2009), 379–474.

[9] Gao Cong, Wendin Fan, Florio Geeris, Jianzhong Li, and Jiahao Luo. 2012. On the Complexity of View Update Analysis and Its Application to Annotation Propagation. *IEEE TKDE* 24, 3 (2012), 586–519. https://doi.org/10.1109/TKDE.2011.27

[10] Stergos S. Cosmadakis and Christos H. Papadimitriou. 1984. Updates of Relational Views. *J. ACM* 31, 4 (Sept. 1984), 742–760. https://doi.org/10.1145/64836.64838

[11] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. 2000. Tracing the lineage of view data in a warehousing environment. *ACM TODS* 25, 2 (2000), 179–227.

[12] Umeshwar Dayal and Philip A. Bernstein. 1982. On the Correct Translation of Update Operations on Relational Views. *ACM TODS* 7, 3 (1982), 381–416. https://doi.org/10.1145/319702.319740

[13] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi. 1995. *The Semantics of Updates in Relational Databases*. In *STOC* 1995. ACM, New York, NY, USA, 77–90. https://doi.org/10.1145/800105.803397

[14] W. W. Cohen, J. Dean, M. Langheinrich, D. elevation, and T. Suel. 2013. On the Complexity of Consistent Query Answering for Atoms with Simple Keys. In *PODS* 2013. 45–56. https://doi.org/10.1145/2488108.2488148

---

Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Melisou
A DETAILED PROOFS

A.1 Proofs for Section 3.3

Proof of Proposition 12. We first argue that R-tuples are not the optimal choice for a contingency set. Let Γ be a minimum contingency set containing tuple (1, 2).

Case 1: D contains only A(1) or C(1) but not both. WLOG, suppose it contains only A(1). We can then obtain a contingency set Γ’ = (Γ - R(1, 2)) ∪ A(1) of size k. Similar if it contains only C(1).

Case 2: D contains both A(1) and C(1). Consider Γ'' = (Γ ∪ A(1)) - R(1, 2) and Γ''' = (Γ ∪ C(1)) - R(1, 2), and suppose that neither of those is a contingency set. Then we have A(i), R(i, 2), R(1, 2), C(1) in D - Γ’ and A(1), R(1, 2), R(j, 2), C(j) in D - Γ’’. However, the existence of those joins implies that D - Γ has the join A(i), R(i, 2), R(j, 2), C(j) contradicting the fact that Γ is a contingency set. Therefore, at least one of Γ’, Γ’’ must be a contingency set and we can replace R(1, 2) by A(1) or C(1).

Since R can be made exogenous, solving resilience for this query is the same as solving vertex cover in a bipartite graph, and therefore is in P.

A.2 Proofs for Section 4.2

Proof of Lemma 14. First observe that disconnected components join as a cross-product, so for a query to be made false it is enough that at least one of its query components is made false. Hence, for each query component q_i, if D - Γ_i ≠ q_i, then D - Γ_i ≠ q, which then implies ρ(q, D) = min_i ρ(q_i, D).

Proof of Lemma 15. Suppose that q is not minimal, q’ is its minimal form, and q_i is a query component in q but not in q’. This implies that there exists a component q_j in q, such that q_j ⊈ q_i. Therefore, D ⊈ q_j → D ⊈ q_j and we can conclude that any Γ_j is also a contingency set for D, q_j. This guarantees that min_i Γ_i is not going to be the min’ for the entire query, since any Γ_i is going to have same size or be smaller than min Γ_i. Note that the complexity of q_j doesn’t influence the complexity of q, since we don’t need to solve RES(q_j). Therefore, we must assume the query is minimal in order to obtain the hardness result.

A.3 Proofs for Section 4.3

Proof of Proposition 18. We show that tuples from dominated relations don’t need to be used in minimum contingency sets. Assume q is a connected query and let Γ be a minimum contingency set of q in D.

Suppose that relation A dominates relation B and there is some tuple B(t) that is in Γ. Tuple B(t) can participate in joins as one or more of the B-atoms in q. Let’s call those atoms B_i, for i ∈ [k]. Our definition of domination guarantees that there exists an atom A_j for each atom B_i such that the projection of t onto var(A_j) always produces the same tuple p. Then we can replace B(t) by A(p) and we remove at least as many witnesses if D |= q.

For disconnected queries, suppose that A dominates relation B for each component independently. By Lemma 14, to compute Γ for q we just need to compute Γ_i for each component and take the minimum, and since B is dominated in each component, the same argument for connected components holds here, and we can make B exogenous in q.

As a result we show the complexity of RES(q) is the same if B is made exogenous and therefore RES(q) = RES(q’).

A.4 Proofs for Section 5.1

Lemma 46. RES(q_{rats}) and RES(q_{rats}) are NP-complete.

Proof of Lemma 46. We first show that RES(q_{rats}) is NP-complete by a reduction from 3SAT, similar to the one used to prove RES(q_c) is NP-complete (Proposition 51).

Let ψ be a 3CNF formula with n variables v_1, . . . , v_n and m clauses C_0, . . . , C_m−1. Our reduction will map any such ψ to a pair (D_ψ, k_ψ) where D_ψ is a database satisfying q_{rats}, and ψ ∈ 3SAT ⇔ (D_ψ, k_ψ) ∈ RES(q_{rats}).

In our construction, if ψ ∈ 3SAT, then the size of each minimum contingency set for q_{rats} in D_ψ will be k_ψ = 6mn, whereas if ψ /∈ 3SAT, then the size of all contingency sets for q_{rats} in D_ψ will be greater than k_ψ.

We construct D_ψ by taking D_ψ from the proof of Proposition 51, and adding the following tuples for each join (a, b, c) in D_ψ, q_△:

R = {(a, b), (b, c), (c, a)}
A = {(a), (b), (c)}

Notice that for each join (a, b, c) in D_ψ we thus create 3 joins, (a, b, c), (b, c, a), (c, a, b) in D_ψ, but they all use the same R-triangle.

We know from Proposition 51 that some R-tuples participate in 2 joins (triangles) and some only in 1 within a variable gadget. Thus, in D_ψ these numbers are 6 joins or 3 joins. Observe that A-tuples participate in at most 2 joins each, so it is never better to choose an A-tuple instead of an R tuple. Therefore it follows that the same choice of tuples for the minimum contingency set for D_ψ, q_△ will also work for D_ψ, q_{rats} by choosing the corresponding R-tuples in D_ψ based on the R, S, T-tuples chosen from D_ψ.

For q_{rats} the reduction is similar, but the final atom – R(x, z) instead of R(z, x) – must be handled. The solution is that for each join (a, b, c) in D_ψ, q_△, we add the following tuples to D_ψ:

R = {(a, b), (b, a), (b, c), (c, b), (c, a), (a, c)}
A = {(a), (b), (c)}

Now, each join from D_ψ, q_△ leads to 6 joins in D_ψ, q_{rats} – the three from the above proof plus their reversals. Thus, the R-tuples for solid edges from Figure 16 are used in 6 joins each, whereas A-tuples are in at most 4 joins each. Thus, based on the minimum contingency sets for D_ψ, q_△, we create minimum contingency sets for D_ψ, q_{rats} by including the corresponding R-tuples and their reversals.

Lemma 47. RES(q_{rats}), RES(q_{rats}) and RES(q_{rats}) are NP-complete.

Proof of Lemma 47. The same idea used above to prove that RES(q_{rats}) is hard, works for query q_{rats}. When defining D_ψ for
this case, we just need to add the appropriate $B$-tuples:
\[
R = \{(a, b), (b, c), (c, a)\}
\]
\[
A = \{(a), (b), (c)\}
\]
\[
B = \{(a), (b), (c)\}
\]
Since $B$-tuples have the same properties as the $A$-tuples, they are never better choices than $R$-tuples and we can obtain a minimum contingency set with only $R$-tuples, as we saw in Lemma 46 above. Similar reduction thus follow for $\text{RES}(q_{\text{rats}}^{\text{rats}})$ and $\text{RES}(q_{\text{rats}}^{\text{brels}})$.

\section{A.5 Proofs for Section 6}

\textbf{Proof of Theorem 24 (Unary Path).} We define a reduction from $\text{RES}(q_{\text{vc}})$. Given a database $D$, we want to define a database $D'$ such that
\[
(D, k) \in \text{RES}(q_{\text{vc}}) \iff (D', k) \in \text{RES}(q) \tag{1}
\]
We can assume that $A(x)$ and $A(y)$ are consecutive occurrences of $A$ so let $p$ be a subquery of $q$ consisting of a path from $A(x)$ to $A(y)$ with no intervening occurrences of $A$. Thus, $q = q_{\text{path}}A(x)pA(y)q_{\text{path}}$. Since $A$ is the only $s_j$ relation, the relations that occur in $p$ occur only in $p$.

For each atom $R_i(v_1, v_2)$ occurring in $p$, we define
\[
R_i' = \{(t(v_1, a, b), t(v_2, a, b)) \mid D \models q_{\text{vc}}(a, b)\}
\]
where
\[
t(v, a, b) \overset{\text{def}}{=} \begin{cases} a & \text{if } v = x \\ b & \text{if } v = y \\ \langle ab \rangle_v & \text{otherwise} \end{cases}
\]
In other words, $x$ maps to $a$, $y$ maps to $b$, and any other variable $v$ maps to $\langle ab \rangle_v$. Thus, we have made a faithful copy of $D$ capturing $q_{\text{vc}}$. For the other atoms, $S_j(v_1, v_2)$, not in $p$, let
\[
S_j' = \{(m(v_1, a, b), m(v_2, a, b)) \mid D \models q_{\text{vc}}(a, b)\}
\]
where $m(v, a, b)$ matches with $t(v, a, b)$ as well as with a set of $n$ new values, where $n = |\text{dom}(D)|$. It follows that there is always a minimum contingency set for $(D', q)$ with only $A$-tuples, in particular, the sets $\{A(a) \mid V(a) \in \Gamma\}$ for any minimum contingency set for $(D, q_{\text{vc}})$.

\textbf{Proof of Theorem 25 (Binary Path).} Similar to the unary case, we define a reduction from $\text{RES}(q_{\text{vc}})$. Given a database $D$ we want to define a database $D'$ such that
\[
(D, k) \in \text{RES}(q_{\text{vc}}) \iff (D', k) \in \text{RES}(q) \tag{2}
\]
Consider $q = q_TR(x, y)pR(z, w)q_T$, and that $p$ is a subquery of $q$ consisting of a path from $R(x, y)$ to $R(z, w)$ without intervening occurrences of $R$. By assumption, there is no path of just $R$’s from $R(x, y)$ to $R(z, w)$, so we may assume that $R(x, y)$ and $R(z, w)$ have such an $R$-free path, $p$, between them.

In order to define the reduction, we define an equivalence relation, $\equiv$, on the variables occurring in $q$, namely $u \equiv v$ iff $q$ has an $R$-path from $u$ to $v$, i.e., there is a path of $R$-atoms occurring in $q$ that takes us from $u$ to $v$. (For example, for the query $R(x, y), S(u, z), R(z, w), Q(w, x), R(x, v)$, the equivalence classes of $\equiv$ are $\{x, y\}$, $\{z\}$, $\{u\}$.) Note that by assumption, for the equivalence relation defined by $q, x \not\equiv z$.

For any atom $S_i(v_1, v_2)$ occurring in $R(x, y)pR(z, w)$, we define
\[
S_i' = \{(t'(v_1, a, b), t'(v_2, a, b)) \mid D \models q_{\text{vc}}(a, b)\}
\]
where
\[
t'(v, a, b) \overset{\text{def}}{=} \begin{cases} a & \text{if } v \equiv x \\ b & \text{if } v \equiv z \\ \langle ab \rangle_v & \text{otherwise} \end{cases}
\]
Additionally, for atoms $T_j(v_1, v_2)$ occurring in $q_T$, let
\[
T_j' = \{(m(v_1, a, b), m(v_2, a, b)) \mid D \models q_{\text{vc}}(a, b)\}
\]
where $m(v, a, b)$ matches with $t'(v, a, b)$ as well as with a set of $n$ new values, where $n = |\text{dom}(D)|$.

We have that all $R$-tuples in $D'$ will have the same value as first and second attributes, so $R$ can be seen as corresponding to relation $A$ in $D$. Similar to the unary case, we have made a copy of $D$ capturing $q_{\text{vc}}$ and there is always a minimum contingency sets for $(D', q)$ with only $R$-tuples, in particular, the sets $\{R(a, a) \mid V(a) \in \Gamma\}$ for any minimum contingency set for $(D, q_{\text{vc}})$. 

\section{Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou
A.6 Proofs for Section 7.1
These are the variations of \( q_{\text{chain}} \) with unary relations:
\[
\begin{align*}
q_{\text{chain}}^a &\Rightarrow A(x), R(x, y), R(y, z) & \text{Lemma 49} \\
q_{\text{chain}}^b &\Rightarrow R(x, y), B(y), R(y, z) & \text{Lemma 48} \\
q_{\text{chain}}^c &\Rightarrow R(x, y), R(y, z), C(z) & \text{Lemma 49} \\
q_{\text{chain}}^{ab} &\Rightarrow A(x), R(x, y), B(y), R(y, z) & \text{Lemma 49} \\
q_{\text{chain}}^{bc} &\Rightarrow A(x), R(x, y), R(y, z), C(z) & \text{Lemma 49} \\
q_{\text{chain}}^{abc} &\Rightarrow A(x), R(x, y), B(y), R(y, z), C(z) & \text{Lemma 50} \\
\end{align*}
\]
We next show all of them are hard queries.

Lemma 48. \( \text{RES}(q_{\text{chain}}^b) \) is NP-complete.

Proof of Lemma 48. For this case we are going to use almost the same reduction as the one used for \( \text{RES}(q_{\text{chain}}) \), just with the added \( B \)-tuples. Then we argue that there is always a min \( \Gamma \) that only uses \( R \)-tuples.

Let \( \psi \) be a 3CNF formula with \( n \) variables \( v_1, \ldots, v_q \) and \( m \) clauses \( C_1, \ldots, C_m \). Our reduction will map any such \( \psi \) to a pair \((D_\psi, k_\psi)\) where \( D_\psi \) is a database satisfying \( q_{\text{chain}}^b \) and
\[
\psi \in \text{3SAT} \iff (D_\psi, k_\psi) \in \text{RES}(q_{\text{chain}}^b)
\]
In our construction, if \( \psi \in \text{3SAT} \), then the size of each minimum contingency set for \( q_{\text{chain}}^b \) in \( D_\psi \) will be \( k_\psi = (n + 5)m \), whereas if \( \psi \notin \text{3SAT} \), then the size of all contingency sets for \( q_{\text{chain}}^b \) in \( D_\psi \) will be greater than \( k_\psi \).

First, include in \( D_\psi \) all the same \( R \)-tuples included in the proof of Proposition 10. In addition to that add the following \( B \)-tuples:

(1) Variable gadget: For each variable \( v_i \) and each \( j \in [m] \) insert the following two tuples into the database: \( B(v'_i, j) \) and \( B(v''_i, j) \).

(2) Clause gadget: For each clause \( j \in [m] \) insert the following 6 tuples into the database: \( B(a_j), B(b_j), B(c_j), B(a'_j), B(b'_j), B(c'_j) \).

By adding those tuples, we obtain the same structure and joins of the reduction for \( \text{RES}(q_{\text{chain}}) \). Now suppose that \( t = B(d) \) is in a minimum contingency set \( \Gamma \). If \( d = v'_i \) or \( v''_i \) for some \( i, j \), we know that \( t \) must join with \( t' = R(v''_i, j) \) (or \( R(v'_i, j) \)) by our construction. Thus, we can exchange \( t \) for \( t' \) and obtain contingency set \( \Gamma' \). Similar, if \( d \in \{a, b, c, a', b', c'\} \), then \( t \) must join with tuple \( R(d, *) \), since there is only tuple of that kind for each possible value of \( d \).

This shows that there is a minimum contingency set for \( D_\psi \) without \( B \)-tuples, and the properties of the reduction in Proposition 10 also hold in this case.

Lemma 49. \( \text{RES}(q_{\text{chain}}^a) \), \( \text{RES}(q_{\text{chain}}^c) \), \( \text{RES}(q_{\text{chain}}^{ab}) \) and \( \text{RES}(q_{\text{chain}}^{bc}) \) are NP-complete.

Proof of Lemma 49. We again define a reduction from 3SAT, using gadgets similar to the one in Proposition 10. The variable gadget remains such that a minimum cover will choose either blue nodes (variable is set to true), or red nodes (variable is set to false). The clause gadget (black nodes) is chosen as to enforce a clause:
if one or more of the outermost black nodes are chosen, then the minimum cover is 5, otherwise 6.

We next reduce 3SAT to \( \text{RES}(q_{\text{chain}}^a) \). Let \( \psi \) be a 3CNF formula with \( n \) variables \( v_1, \ldots, v_q \) and \( m \) clauses \( C_1, \ldots, C_m \). Our reduction will map any such \( \psi \) to a pair \((D_\psi, k_\psi)\) where \( D_\psi \) is a database satisfying \( q_{\text{chain}}^a \) and
\[
\psi \in \text{3SAT} \iff (D_\psi, k_\psi) \in \text{RES}(q_{\text{chain}}^a)
\]
In our construction, if \( \psi \in \text{3SAT} \), then the size of each minimum contingency set for \( q_{\text{chain}}^a \) in \( D_\psi \) will be \( k_\psi = (n + 5)m \), whereas if \( \psi \notin \text{3SAT} \), then the size of all contingency sets for \( q_{\text{chain}}^a \) in \( D_\psi \) will be greater than \( k_\psi \).

(1) Variable gadget: For each variable \( v_i \) and each \( j \in [m] \) insert the following tuples into the database: \( R(v'_i, j), R(v''_i, j') \) and \( A(v'_i), A(v''_i) \). If \( j + 1 > m \), then make the superscript 1. The resulting joins between the tuples form a cycle of length 2m. The minimum contingency sets are to either choose all tuples \( R(v'_i, j) \) representing a variable to have assignment \( \text{true} \), or all tuples \( R(v'_i, j'), A(v'_i) \) representing a variable to have assignment \( \text{false} \). Note that any \( A \)-tuple only joins once, therefore it is better to choose an \( R \)-tuple, since all of these join at least twice.

(2) Clause gadget: For each clause \( j \in [m] \) insert the following tuples into the database: \( B(a_j), B(b_j), B(c_j), A(a'_j), A(b'_j), R(b''_j, j') \). The resulting joins form a triangle. If either of the \( R(*', *) \) is removed, then the remaining joins can be destroyed by choosing only 2 or more tuples, otherwise we need 3. Similar to the variable gadget, \( A \)-tuples are not an optimal choice because they only participate in one join each.

(3) Connecting the gadgets: For each variable \( i \) that appears in clause \( j \) at position 1, add the following tuples: \( R(a''_i', a'_j) \) and \( A(a''_i') \). If \( v_i \) appears as positive add tuple \( R(a''_i', v''_i') \), if it appear as negative add tuple \( R(a''_i', v'_i') \). Analogously use \( b''_i', b'_i', c''_i', c'_i' \) instead of \( a''_i', a'_j \) for positions 2 and 3 instead of position 1.

Observe that if the clause is not satisfied, then we need to choose the \( A \)-tuples (orange squares in Fig. 12), and not choose the outer black nodes (\( R \)-tuples) in the clause gadget, resulting in choosing 6 tuples in total in order to delete all the joins, otherwise we just need 5 tuples.

The reduction for \( q_{\text{chain}}^{abc} \) is similar to the one presented above. First, use the same \( D \) just adding the appropriate \( B \)-tuples, i.e., \( B \)-tuples that preserve the joins.

Now note that for any \( t = B(d) \) in \( D_\psi \), there is only one \( R \)-tuple such that \( t' = R(d, *) \), therefore \( t \) must join with \( t' \). Therefore, any occurrence of \( B \)-tuple in a contingency set can be exchanged by its correspondent \( R \)-tuple, and we are guaranteed this reduction has the same properties as the one for \( q_{\text{chain}}^a \).

\]

Lemma 50. \( \text{RES}(q_{\text{chain}}^{bc}) \) and \( \text{RES}(q_{\text{chain}}^{abc}) \) are NP-complete.
Proof of Lemma 50. We define a reduction from 3SAT. As in the previous cases, the variable gadget remains such that a minimum cover will choose either blue nodes (variable is set to true), or red nodes (variable is set to false). The clause gadget (center black nodes) is chosen as to enforce a clause: if one or more of the outermost joins (black edges) are deleted by choosing the corresponding A-tuple (orange square), then the minimum cover for the black subgraph is 2, otherwise 3.

We next reduce 3SAT to RES(qac). Let \( \psi \) be a 3CNF formula with \( n \) variables \( v_1, \ldots, v_n \) and \( m \) clauses \( C_1, \ldots, C_m \). Our reduction will map any such \( \psi \) to a pair \((D_\psi, k_\psi)\) where \( D_\psi \) is a database satisfying \( q_{\text{chain}}^\text{ac} \), and

\[
\psi \in 3\text{SAT} \iff (D_\psi, k_\psi) \in \text{RES}(q_{\text{chain}}^\text{ac})
\]

In our construction, if \( \psi \in 3\text{SAT} \), then the size of each minimum contingency set for \( q_{\text{chain}}^\text{ac} \) in \( D_\psi \) will be \( k_\psi = (n + 5)m \), whereas if \( \psi \notin 3\text{SAT} \), then the size of all contingency sets for \( q_{\text{chain}}^\text{ac} \) in \( D_\psi \) will be greater than \( k_\psi \).

1. Variable gadget: For each variable \( v_i \) and each \( j \in [m] \) insert the following tuples into the database: \( R(v'_i, v''_i) \), \( R(v'_i, v''_i^{+1}) \) and \( A(v'_i), A(v''_i) \) and \( C(v'_i), C(v''_i) \). If \( j + 1 > m \), then make the superscript 1. The resulting joins between the tuples form a cycle of length \( 2m \). The minimum contingency sets are to either choose all tuples \( R(v'_i, v''_i) \) representing a variable to have assignment true, or all tuples \( R(v'_i, v''_i^{+1}) \) representing a variable to have assignment false. If we only consider those tuples, note that A- and C-tuples participate in only one join, so the optimal choice is to delete R-tuples.

2. Clause gadget: For each clause \( j \in [m] \) insert the following tuples into the database: \( R(a_j, b_j) \), \( R(b_j, c_j) \), \( R(c_j, a_j) \), \( R(a'_j, a_j) \), \( R(b'_j, b_j) \), \( R(c'_j, c_j) \), \( A(a_j) \), \( A(b_j) \), \( A(c_j) \), \( A(a'_j), A(b'_j), A(c'_j) \), \( C(a_j), C(b_j), C(c_j) \). The resulting joins form a triangle. If either of the \( A(a'_j) \) is removed, then the remaining joins can be destroyed by choosing only 2 or more tuples, otherwise we need 3. We later argue that these tuples only need be R-tuples.

3. Connecting the gadgets: For each variable \( i \) that appears in clause \( j \) at position 1, add the following tuples: \( R(a'_j, a''_j) \), \( R(a''_j, a'_j') \), and \( C(a''_j) \). If \( v_i \) appears as positive add tuple \( R(v'_i, a'_j) \), if it appear as negative add tuple \( R(v'_i, a''_j) \). Analogously use
Choosing all the nodes chosen for a minimum contingency set, as well as green nodes.

Figure 12: Excerpt from the construct showing the gadget for clause $C_1 = (v_1 \lor v_2 \lor v_3)$. We omit the $A$-tuples that participate in only one join, since they shall never be chosen for a minimum contingency set, as well as green nodes.

Figure 13: Excerpt from the construct showing the gadget for clause $C_1 = (v_1 \lor v_2 \lor v_3)$. We omit the $A$-tuples and $C$-tuples that would not be chosen for a minimum contingency set, as well as green nodes.

$b_j', b_j''$ or $c_j', c_j''$ instead of $a_j', a_j''$ for positions 2 and 3 instead of position 1.

With our gadget, if the clause cannot be satisfied, then we need to choose all the $C$-tuples (orange diamonds on Fig. 13), since we can delete two joins by doing deleting each. In that case, in order to delete the remaining joins we need to delete 3 tuples, namely the 3 black nodes in the triangle, resulting on the total deletion of 6 tuples.
We now need to argue that, besides the tuples depicted in Fig. 13, we don’t need other $A$- or $C$-tuples for a minimum contingency set. Assume there is a tuple $t = (A(d))$ in a min $\Gamma$. Given that $d \notin \{a_j', b_j', c_j'\}$, our construction guarantees there is only one $R$-tuple such that $t' = (R(d,-))$, therefore we can have $\Gamma' = \Gamma - t + t'$, and $\Gamma'$ is also a minimum contingency set. Similarly, if there is a tuple $t = (C(d))$ in $\Gamma$, and assuming $d \notin \{a_j', b_j', c_j'\}$, there is only one $R$-tuple $t' = (R(-,d))$, and therefore the same follows.

For $q_{\text{chain}}$ use almost the same construction as above. We just add the appropriate $B$-tuples and show that there is a minimum contingency set that does not contain those.

Consider $D_\psi$ as initially defined for $q_{\text{chain}}$. Now we include the appropriate $B$-tuples:

1. Variable gadget: For each variable $v_i$ and each $j \in [m]$ insert the following tuples into the database: $B(v_i^j)$, $B(v_i^j)$
2. Clause gadget: For each clause $c_j$ insert the following tuples into the database: $B(a_j)$, $B(b_j)$, $B(c_j)$.
3. Connecting the gadgets: For each variable $i$ that appears in clause $j$ at position 1, add tuple $B(a_i^j)$. Analogously, add $B(b_i^j)$ and $B(c_i^j)$ for positions 2 and 3, respectively.

By adding those $B$-tuples we obtain the same joins we saw in the reduction for $\text{RES}(q_{\text{chain}})$. With this construction we guarantee that for any tuple $t = B(d)$, there is either only one tuple $R(d,-)$ or only one tuple $R(-,d)$, which means we can always choose one of those $R$-tuples instead and obtain another minimum contingency set without $B$-tuples.

Proof of Proposition 27. Suppose that $R(x, y), R(y, z)$ are the unique $R$-atoms in $q$. Assume first that there are no unary atoms $A(x), B(y), C(z)$. We define a reduction from $\text{RES}(q_{\text{chain}})$ to $\text{RES}(q)$ as follows:

Consider a database $D$ with $D \models q_{\text{chain}}$ and we may assume that there are no loops $R(a, a) \in D$, since those would have to be in any $\Gamma$. We define a new database $D'$ such that for each atom $S_i(t_1, t_2)$ or $A(t)$ occurring in $q$, we define

\[
S_i = \{(t(a, b, c), t(v, a, b, c)) \mid D \models q_{\text{chain}}(a, b, c)\}
\]

\[
A = \{(t(a, b, c)) \mid D \models q_{\text{chain}}(a, b, c)\}
\]

where

\[
t(v, a, b) = \begin{cases} 
    a & \text{if } v = x \\
    b & \text{if } v = y \\
    c & \text{if } v = z \\
\{a, b, c\}_v & \text{otherwise.}
\end{cases}
\]

Now we want to show

\[
(D, k) \in \text{RES}(q_{\text{chain}}) \iff (D', k) \in \text{RES}(q)
\]

Notice that this mapping from $D$ to $D'$ preserves the joins in $D, q_{\text{chain}}$. Moreover, there are no new joins created where variables $x, y, z$ are mapped to values that did not correspond to joins before. Since $q$ is pseudo-linear, no endogenous atom of $q$ contains both $x$ and $z$. Therefore, any minimum contingency set for $D, q_{\text{chain}}$ is also a minimum contingency set for $D', q$. This completes our reduction.

Now, if any subset of unary relations $A(x), B(y), C(z)$ does appear in $q$, then we define a reduction from the appropriate unary variation of $q_{\text{chain}}$. The same mapping used above to define $D'$ from $D$ preserves all minimum contingency sets, as desired.

A.7 Proofs for Section 7.3

Proof of Proposition 31. We define a reduction from 3SAT to $\text{RES}(q_{\text{perm}}^{AB})$, see Figure 14. Similar to the previous cases, we want to create variable gadgets such that a minimum cover will choose either blue nodes (variable is set to true), or red nodes (variable is set to false), and a clause gadget (black nodes) such that if the clause is satisfied, then the minimum cover is 5, otherwise 6.

Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_1, \ldots, C_m$. Our reduction will map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $\pi_\psi,$ and

\[
\psi \in \text{3SAT} \iff (D_\psi, k_\psi) \in \text{RES}(q_{\text{perm}}^{AB})
\]

In our construction, if $\psi \in \text{3SAT},$ then the size of each minimum contingency set for $q_{\text{perm}}^{AB}$ in $D_\psi$ will be $k_\psi = (3n + 5)m,$ whereas if $\psi \notin \text{3SAT}$, then the size of all contingency sets for $q_{\text{perm}}^{AB}$ in $D_\psi$ will be greater than $k_\psi.$

(1) Variable gadget: For each variable $v_i$ and each $j \in [m]$ insert the following tuples into the database: $A(v_i^j, B(v_i^j), A(v_i^j)$, $B(v_i^j)$ and $R(v_i^j, v_i^j), R(v_i^j, v_i^j), R(v_i^j, v_i^j), R(v_i^j, v_i^j), R(v_i^j, v_i^j), R(v_i^j, v_i^j), R(v_i^j, v_i^j), R(v_i^j, v_i^j))$. If $j + 1 > m$, then make the superscript 1.

We want to join those tuples such that the minimum contingency sets are to either choose all tuples $A(v_i^j, B(v_i^j), A(v_i^j)$ representing a variable to have assignment $\text{true}$, or all tuples $A(v_i^j, B(v_i^j), A(v_i^j)$ representing a variable to have assignment $\text{false}$, plus some $R$-tuples. To obtain that property, we need the following additional tuples: $A(\pi_\psi, B(\pi_\psi), A(\pi_\psi)$ and $R(\pi_\psi, v_i^j), R(\pi_\psi, v_i^j), R(\pi_\psi, v_i^j), R(\pi_\psi, v_i^j), R(\pi_\psi, v_i^j))$. With this construction we guarantee that we can "cover" the variable gadget by choosing either all positive $A$, $B$-tuples plus the $m$ tuples $R(\pi_\psi, v_i^j), v_i^j), or all negative $A, B$-tuples plus the $m$ tuples $R(\pi_\psi, v_i^j), v_i^j). In both cases, we choose 3m tuples.

(2) Clause gadget: For each clause $c_j$ insert the following tuples into the database: $A(a_j)$, $B(a_j)$, $B(b_j)$, $B(c_j)$, $R(a_j, b_j)$, $R(b_j, a_j)$, $R(c_j, a_j)$, $R(a_j, c_j)$, $R(b_j, c_j)$, $R(c_j, b_j)$, $R(c_j, a_j)$, $R(a_j, b_j)$, $R(b_j, c_j)$, $R(c_j, b_j)$, $R(c_j, a_j)$ and $A(a_j)$. Analogously use $b_j$ or $c_j$ instead of $a_j$ for positions 2 and 3 instead of position 1.

After connecting the variable gadgets with the clause gadgets, the joins are formed such that if a clause cannot be satisfied, then we need to pick all $A$- and $B$-tuples from the clause gadget (the black triangle), totaling 6 tuples. Otherwise, we can delete all joins by picking 3 tuples, namely 2 pairs of $A, B$-tuples and one $R$-tuple.

Proof of Proposition 32. There are 2 cases.

Case 1: $q$ is pseudo-linear and not bound. We can write $q = q_\ell(x), G(x, y)$ where $q_\ell(x)$ does not contain the variable $y$. $G(x, y)$...
includes \(R(x, y), R(y, x)\) and may include exogenous atoms containing the variable \(y\). Think of \(G(x, y)\) as the rightmost group in Figure 5.

For any database, \(D \models q\), \(\text{RES}(D, q)\) is equivalent to the following Network Flow. As usual, each endogenous atom from the pseudo-linear \(q_x(x)\) becomes a 1-weight edge and each exogenous atom is an \(\infty\)-weight edge. Whenever \(\{R(c, d), R(d, c)\} \subseteq D\), we add \(\infty\)-weight edges from the rightmost output of \(q_x(c)\) and \(q_x(d)\) to \(\{c, d\}\) and a 1-weight edge from \(\{c, d\}\) to the terminal node, \(t\).

Case 2: \(q\) is bound. We can write \(q = q_x(x), G(x, y), q_x(y)\) where \(G(x, y)\) includes \(R(x, y), R(y, x)\) and may include an essentially exogenous atom \(D(x, y)\) if that occurs in \(q\). The relevant issues are that removing \(G(x, y)\) separates \(q_x(x)\) from \(q_x(y)\) and these contain at least one endogenous atom each.

We define a reduction from \(\text{RES}(q_{\alpha B})\) to \(\text{RES}(q)\). We say that variable \(z\) \(\text{isLike}\) \(x\), if \(z\) occurs in \(q_x(x)\). Otherwise, \(z\) \(\text{isLike}\) \(y\).

Now consider a database \(D\) with \(D \models q_{\alpha B}\). We define a new database \(D'\) such that for each atom \(S_v(v_1, v_2)\) in \(q\), we define

\[
S' = \{ (t(v_1, a, b), t(v_2, a, b)) \mid D \models q_{\alpha B}(a, b) \}
A' = \{ (t(v, a, b)) \mid D \models q_{\alpha B}(a, b) \}
\]

where

\[
t(v, a, b) = \begin{cases} a & \text{if } v \text{ \(\text{isLike}\) } x \\
 b & \text{if } v \text{ \(\text{isLike}\) } y 
\end{cases}
\]

It is clear that the joins and minimum contingency sets of \(D \models q_{\alpha B}\) are exactly preserved in \(D' \models q\).

\section{Proofs for Section 8.1}

\textbf{Proof of Proposition 36.} We define a reduction from \(\text{RES}(q_{\alpha B})\) to \(\text{RES}(q)\), using a strategy similar to the proof of Theorem 24.

\section{Proofs for Section 8.2}

\textbf{Proof of Proposition 37.} We reduce Max 2-SAT to \(\text{RES}(q_{\alpha C})\). Given a 2CNF formula, \(\varphi\), with \(n\) variables and \(m\) clauses, and a number \(r < m\), we produce a database, \(D\), and bound \(k\), such that \(\varphi\) has an assignment satisfying at least \(r\) clauses iff \((D, k) \in \text{RES}(q_{\alpha C})\). The construction is drawn in Figure 5. A sample variable gadget for variable \(x\) is shown. The two minimum contingency sets consist of 2\(x\) \(x\) nodes, plus 2 helper nodes in the two crossover gadgets or 2\(x\) \(\bar{x}\) nodes, plus 2 helper nodes, corresponding to variable \(x\) being true or false, respectively. The reason for the crossover is so that each variable can be instantiated via diamonds and hexagons corresponding to the atoms \(A, C\), respectively.

The clause gadgets for clauses of size 1 and size 2 are also drawn. Clauses of size 1 need no nodes chosen when they are true and one node otherwise. Clauses of size 2 need 1 node chosen when they are true and 2 when they are false. Let \(d\) be the number of clauses of size 2 in \(\varphi\). Saying that at least \(r\) clauses of \(\varphi\) are true means that at most \(m - r\) clauses are false. Thus, the size of the minimum contingency set is \(k = n(2s + 2) + d + m - r\).

\textbf{Proof of Proposition 39.} First observe that any contingency set contains only \(R\)-tuples, since \(S, T\) are dominated and therefore exogenous. For any tuple \(R(a, b) \in D\), if \(S(a, b), T(a, b) \in D\), then \(R(a, b)\) must be in all contingency sets, since those 3 tuples form a join. Let \(\Gamma\) be the set of all such tuples. We then proceed to create
a flow with tuples $D' = D - \Gamma_T S$ and we claim that $\Gamma = \Gamma_T S \cup C$ is a min contingency set for $(q_{3\text{cond}}^A, D)$, where $C$ is a min cut found by FLOW.

Let $C$ be a min cut and suppose there is a $\Gamma$ such that $D' = \Gamma \setminus q_{3\text{cond}}^A$ and $|C| > |\Gamma'|$. That implies that there are at least 2 joins that can be broken by deleting one tuple but the min cut chose to delete 2 edges. Consider the tuple $R(a, b)$ and these joins to be

$T(a, b)R(a, b)R(1, b) \in \{\Sigma, 1\}$

$T(3, b)R(3, b)RaR(a, 4) \in \{\Sigma, 1\}$

Note that with this set of tuples we also have join

$T(3, b)R(3, b)R(1, b) \in \{\Sigma, 1\}$

which cannot be deleted by deleting $R(a, b)$, contradicting the assumption that it was possible.

A.10 Proofs for Section 8.3

Proof of Proposition 40. Reduction from $\text{RES}(q_{\text{chain}})$. □

Proof of Proposition 41. Reduction from Max 2SAT, similar to the one used for $q_{3\text{cond}}^A$. □

A.11 Proofs for Section 8.4

Proof of Proposition 42. This is similar to Proposition 13. The difference is that while $A(a)$ “dominates” the 1-way tuple $R(a, b)$ in $q_{2\text{perm}}^A$, it is not the case that $S(ei, a)$ would dominate $R(a, b)$ because there might be many $ei$’s such that $S(ei, a) \in D$, in which case it might be advantageous to choose one $R(a, b)$ instead of many $S(ei, a)$’s.

We thus modify the flow graph to include all the $S(e, a)$ edges at cost 1 each on the left, all the $(a, b)$ pairs at cost 1 each on the right. We include $\infty$-weight edges from any $S(e, a)$ to $(a, b)$ plus cost 1 edges from $S(e, a)$ to $(b, c)$ for any 1-way edges $R(a, b)$.

Let $M$ be a min-cost flow and form $\Gamma$ by including all the $(e, a)$’s and 1-way $R(a, b)$’s from $M$ together with one of $R(a, b)$ or $R(b, a)$ whenever $(a, b) \in M$. Similar to Proposition 13, the rule for which to choose is that some $S(e, a) \in \{D - M\}$ but no $S(f, b) \in \{D - M\}$, then add $R(a, b)$ to $\Gamma$. Symmetrically, if $S(e, b) \in \{D - M\}$ but no $S(f, a) \in \{D - M\}$, then add $R(b, a)$ to $\Gamma$; otherwise, arbitrarily add one or the other.

The same argument as in Proposition 13 shows that $\Gamma$ is a minimum contingency set. □

Proof of Proposition 43. We reduce 3SAT to $\text{RES}(q_{2\text{perm}})$. The idea for the variable gadgets is that for a database that contains the tuples $T_x = \{S(x_i, \overline{x_i}), R(x_i, \overline{x_i}), S(\overline{x_i}, x_i), R(\overline{x_i}, x_i)\}$, we must choose exactly one $R(x_i, \overline{x_i})$ or $R(\overline{x_i}, x_i)$, the first of which will correspond to the assignment $x_i$ to 1, and the second of which, to 0. In full detail, the $x$ gadget consists of a chain of these choices, i.e., the union of $T_{x_i}, i = 1 \ldots m$, together with all the tuples $R(x_i, x_{i+1}), R(x_{i+1}, x_i), R(\overline{x_i}, x_{i+1}), R(\overline{x_{i+1}}, x_i)$. For a minimum contingency over this gadget we may choose all of the $R(x_i, x_{i+1})$ and $R(x_i, \overline{x_i})$ edges (corresponding to $x_i$ gets 1), or all the $R(\overline{x_i}, x_{i+1})$ and $R(x_i, x_i)$ edges (corresponding to $x_i$ gets 0).

The clause gadget is similar. If $C_i$ is $(x \vee \overline{y} \vee z)$, then the clause can eliminate two, but not all three pointers to the edges $\{x_i, x_{i+1}\}$, $\{\overline{y_i}, \overline{y_{i+1}}\}$, $\{z_i, z_{i+1}\}$ after removing 8 tuples. To simplify the explanation, let $P(a, b) = R((a, b), (b, a))$ and $F(a, b) = P(a, b) \cup \{S(a, b), S(b, a)\}$ for elements $a, b \in D$. The $C_i$ clause gadget contains the union of the following sets of tuples: $F(a_i, b_i), F(b_i, c_i), F(c_i, a_i), F(a_i, x_i), F(b_i, \overline{x_i}), F(c_i, z_i), P(a_i, q_i), P(b_i, \overline{q_i}), P(c_i, \overline{q_i})$. The idea is that for each full pair, $F(e, f)$, exactly one of $R(e, f)$ or $R(f, e)$ must be chosen in the minimum contingency set $\Gamma$. $C_i$ is designed so that a contingency set of size 8 exists if at least one pair from $P(x_i, x_{i+1}), P(\overline{y_i}, \overline{y_{i+1}}), P(z_i, z_{i+1})$ has been previously chosen, i.e., if the clause $C_i$ is true. □

Proof of Proposition 44. We reduce $\text{RES}_2(q_{2\text{perm}}^A)$ to $\text{RES}_3(q_{2\text{perm}}^A)$. Given a database $D \models q_{2\text{perm}}^A$, construct $D' \models q_{2\text{perm}}^A$ as

$A' := \{a' \mid A(a) \in D\}$

$R' := R \cup \{\{a', a\} \mid A(a) \in D\}$.

It then follows, that it is always at least as good to put $A(a')$ into $\Gamma$, rather than $R(a')$. Thus, the minimum contingency sets for $(D', q_{2\text{perm}}^A)$ correspond exactly to the minimum contingency sets for $(D, q_{2\text{perm}}^A)$.

For $\text{RES}_3(q_{2\text{perm}}^A)$, Even though $q_{2\text{perm}}^A \rightarrow q_{2\text{perm}}^A$, there is no obvious reduction between $\text{RES}_2(q_{2\text{perm}}^A)$ and $\text{RES}_3(q_{2\text{perm}}^A)$, However, the same reduction from 3SAT to $\text{RES}_2(q_{2\text{perm}}^A)$ in Proposition 31 also works for $\text{RES}_3(q_{2\text{perm}}^A)$.

For $\text{RES}_3(q_{2\text{perm}}^A + 1)$, we can define a reduction from $\text{RES}_3(q_{2\text{perm}}^A)$. □

A.12 Proofs for Section 8.5

Proof of Proposition 45. For $\text{RES}(z_4)$, a reduction from $\text{RES}(q_{\text{INC}})$ is enough. Note that tuples $R(a, b)$ with $a \neq b$ do not need to be in a contingency set.

For $\text{RES}(z_5)$, a reduction from Max 2SAT, similar to the one used in Proposition 37, can be used to show NP-hardness. □

B Relevant Proofs from SJ-FREE Case

Proposition 51 (Triangle $q_{\Delta}$ is hard). $\text{RES}(q_{\Delta})$ is NP-complete.

Proof of Proposition 51. We reduce 3SAT to $\text{RES}(q_{\Delta})$. It will then follow that $\text{RES}(q_{\Delta})$ is NP complete. Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_0, \ldots, C_{m-1}$. Our reduction will map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $q_{\Delta}$, and

$\psi \in 3\text{SAT} \iff (D_\psi, k_\psi) \in \text{RES}(q)$

In our construction, if $\psi \in 3\text{SAT}$, then the size of each minimum contingency set for $q_{\Delta}$ in $D_\psi$ will be $k_{\psi} = 6mn$, whereas if $\psi \notin 3\text{SAT}$, then the size of all contingency sets for $q_{\Delta}$ in $D_\psi$ will be greater than $k_{\psi}$.

Note $D_\psi \models q_{\Delta}$ if it contains three pairs $R(a, b), S(b, c), T(c, a)$. We visualize $R(a, b)$ as a red edge, $S(b, c)$ as a green edge and $T(c, a)$ as a blue edge. Thus each witness $(a, b, c)$ that $D_\psi \models q_{\Delta}$ is an RGB triangle. (Notice that the edge direction $a \rightarrow b$ drawn in Figure 16 corresponds to the variable order in $R$, and analogously for $S$ and $T$.) The job of a contingency set for $q_{\Delta}$ is to remove all RGB triangles.
Gadget for clause \((\neg x \lor \neg y)\)

Gadget for clause \((x)\)

Top half of Variable Gadget

Middle Crossover Part of Variable Gadget

Figure 15: Reduction Gadgets for proof of Proposition 37: diamonds represent \(A\), ellipses, \(R\), and hexagons, \(C\). In the variable gadgets, the minimum contingency sets choose all red vertices and no green, or all green vertices and no red.
\( D_\psi \) contains one circular gadget \( G_i \) for each variable \( v_i \). The circle consists of 12m solid edges, half of them marked \( v_i \) and the other half marked \( \overline{v_i} \) (see Figure 16a and Figure 16b). Note that there are 12m RGB triangles and they can be minimally broken by choosing the 6m \( v_i \) edges or the 6m \( \overline{v_i} \) edges. Any other way would require more edges removed. Thus, each minimum contingency set for \( D_\psi \) corresponds to a truth assignment to the variables of \( \psi \). And there will be a minimum contingency set of size \( k_\psi = 6mn \) iff \( \psi \in 3\text{SAT} \).

We complete the construction of \( D_\psi \) by adding one RGB triangle for each clause \( C_j \). For example, suppose \( C_j = v_1 \lor \overline{v_2} \lor \overline{v_3} \). The RGB triangle we add consists of a red edge marked \( v_1 \), a green edge marked \( \overline{v_2} \) and a blue edge marked \( \overline{v_3} \) (see Figure 16c). Note that if the chosen assignment satisfies \( C_j \), then all \( v_i \) edges are removed, or all \( \overline{v_i} \) edges are removed, or all \( v_i \) edges are removed. Thus the \( C_j \) triangle is automatically removed.

How do we create \( C_j \)'s RGB triangle? Remember that we have chosen \( G_i \) to contain 2 segments for each clause. We use the \( j \)th odd-numbered segment of \( G_i \) to produce the \( v_i \) or \( \overline{v_i} \) used in the clause-\( j \) triangle. The even numbered segments are not used: they serve as buffers to prevent spurious RGB triangles from being created (In Figure 16b we mark these even segments with frowns: they are sad because they are never used).

More precisely, the red \( v_i \)-edge from \( G_1 \) is \((a^1_{j+1}, b^1_{j+1})\), the green \( \overline{v_i} \)-edge from \( G_2 \) is \((b^2_{j+1}, c^2_{j+1})\), and the blue \( \overline{v_i} \)-edge from \( G_3 \) is \((c^3_{j+1}, a^3_{j+1})\) (see Figure 16c).

Now to make this an RGB triangle in \( D_\psi \), we identify the two \( a \)-vertices, the two \( b \) vertices and the two \( c \) vertices. In other words, \( G_1 \)'s \( a \)-vertex \( a^1_{j+1} \) is equal to \( G_3 \)'s \( a \)-vertex \( a^3_{j+1} \), i.e., they are the same element of the domain of \( D_\psi \). We have thus constructed \( C_j \)'s RGB triangle (see Figure 16c).

The key idea is that these identifications can only create this single new RGB triangle because there is no other way to get back to \( G_1 \) from \( G_2 \) in two steps. All other identifications involve different segments and so are at least six steps away. Recall that this is the reason why the odd-numbered segments in the \( G_i \)'s are not used: this ensures that no additional RGB triangles are created.

Thus, as desired, Equation 3 holds and we have reduced 3SAT to \( \text{RES}(q_\alpha) \).

\[ \square \]

**Proposition 52 (Tripod \( \text{qT} \) is hard).** \( \text{RES}(q_\alpha) \) is NP-complete.

**Proof of Proposition 52.** We reduce \( \text{RES}(q_\alpha) \) to \( \text{RES}(q_T) \). It will then follow that \( \text{RES}(q_T) \) is NP-complete. Let \((D, k)\) be an instance of \( \text{RES}(q_\alpha) \). We construct an instance \((D', k)\) of \( \text{RES}(q_T) \) by constructing relations \( A, B, C \) as copies of \( R, S, T \) from \( D \). Define \( D' = (A, B, C) \) as follows:

\[
A = \{(ab) \mid R(a, b) \in D\}
\]

\[
B = \{(bc) \mid S(b, c) \in D\}
\]

\[
C = \{(ca) \mid T(c, a) \in D\}
\]

\[
W = \{((ab), (bc), (ac)) \mid a, b, c \in \text{dom}(D)\}
\]

Here, \( (ab) \) stands for a new unique domain value resulting from the concatenation of domain values \( a \) and \( b \). Observe that there is a 1:1 correspondence between the witnesses of \( D \models q_\alpha \) and

![Gadget construction for hardness proof for \( q_\alpha \).](image-url)

(a) A six-node segment of the gadget \( G_i \). A minimum contingency set chooses either all the solid lines marked \( v_i \), or all the solid lines marked \( \overline{v_i} \). The dotted lines are sad because each of them is only part of one single RGB triangle, thus they are never chosen.

(b) Gadget \( G_i \) is a cycle containing 2m six-node segments with 12m RGB triangles. They can all be eliminated by removing the 6m edges marked \( v_i \) or the 6m edges marked \( \overline{v_i} \). The odd segments are sad because they are never used for connecting different gadgets (corresponding to clauses that use several variables); they only separate the even ones, thus preventing spurious triangles.

(c) For clause \( C_j = (v_1 \lor \overline{v_2} \lor \overline{v_3}) \), we identify vertices \( b^1_{j+1} \in G_1 \) with \( b^2_{j+1} \in G_2 \); \( c^3_{j+1} = G_3 \) and \( a^3_{j+1} = G_1 \). This RGB triangle will be deleted iff the chosen variable assignment satisfies \( C_j \).
the witnesses of \( D' \models q_T \). Thus, every contingency set for \( q_\Delta \) in \( D \) corresponds to a contingency set of the same size for \( q_T \) in \( D' \). Furthermore no minimum \( \Gamma' \) from \( \Gamma' \) needs to choose tuples from \( W \). If \( t = W((ab), (bc), (ac)) \) were in \( \Gamma' \), then we could replace it by \( A(ab) \), which suffices to remove all the witnesses removed by \( t \). As we will explain later, \( A \) “dominates” \( W \) (??). It follows that if \( (D, k) \in \text{RES}(q_\Delta) \) then \( (D', k) \in \text{RES}(q_T) \).

Proof of Lemma 5. Let \( q \) be a query with triad \( T = \{S_0, S_1, S_2\} \). We build a reduction from \( \text{RES}(q_\Delta) \) to \( \text{RES}(q) \). Given any \( D \) that satisfies \( q_\Delta \) we will produce a database \( D' \) that satisfies \( q \) such that for all \( k \):

\[
(D, k) \in \text{RES}(q_\Delta) \iff (D', k) \in \text{RES}(q)
\]  

We will assume that no variable is shared by all three elements of \( T \) (we can ignore any such variable by setting it to a constant). Our proof splits into two cases:

Case 1: \( \text{var}(S_0) \), \( \text{var}(S_1) \), \( \text{var}(S_2) \) are pairwise disjoint. Our reduction is similar to the reduction from \( q_\Delta \) to \( q_T \) (52).

We first define the triad relations in \( D' \):

\[
\begin{align*}
S_0 &= \{(ab, \ldots, ab) \mid R(a, b) \in D\} \\
S_1 &= \{(bc, \ldots, bc) \mid S(b, c) \in D\} \\
S_2 &= \{(ca, \ldots, ca) \mid T(c, a) \in D\}.
\end{align*}
\]

Thus, each tuple of, for example, \( S_0 \) consists of identical entries with value \( (ab) \) for each pair \( R(a, b) \in D \). Thus, \( S_0, S_1, S_2 \) mirror \( R, S, T \), respectively.

To define all the other atoms \( A_i \) of \( D' \), we first partition the variables of \( q \) into 4 disjoint sets: \( \text{var}(q) = \text{var}(S_0) \cup \text{var}(S_1) \cup \text{var}(S_2) \cup V_4 \). Now for each atom \( A_i \), arrange its variables in these four groups. Then define the atom \( A_i' \) of \( D' \) as follows:

\[
A_i' = \{(ab); (bc); (ca); (abc) \mid D \models q_\Delta(a, b, c)\}
\]

For example, all the variables \( v \in \text{var}(S_0) \) are assigned the value \( (ab) \) and all the variables \( v \in V_4 \) are assigned \( (abc) \).

By the definition of triad, there is a path from \( S_0 \) to \( S_1 \) not using any edges (variables) from \( \text{var}(S_2) \). Thus, any witness that \( D' \models q \) which includes occurrences of \( (ab) \) and \( (b'c') \) must have \( b = b' \).

Similarly, a path from \( S_1 \) to \( S_2 \) guarantees that \( c \) is preserved and a path from \( S_2 \) to \( S_0 \) guarantees that \( a \) is preserved. It follows that the witnesses that \( D' \models q \) are essentially identical to the witnesses that \( D \models q_\Delta(x, y, z) \) (See Fig. 17).

Furthermore, any minimum contingency set only needs tuples from \( S_0, S_1 \) or \( S_2 \). For example, if a tuple contains \( (ab) \) or \( (abc) \), then it can be replace by a tuple from \( S_0 \). Thus the sizes of minimum contingency sets are preserved, i.e., Equation 4 holds, as desired. Thus \( \text{RES}(q) \) is NP-complete.

Case 2: \( \text{var}(S_i) \cap \text{var}(S_j) \neq \emptyset \) for some \( i \neq j \): We generalize the construction from Case 1 as follows. Partition \( \text{var}(S_i) \) into those unshared, those shared with \( S_{i-1} \), and those shared with \( S_{i+1} \) (Addition is mod 3).

We then assign the relations of the triad as follows:

\[
\begin{align*}
S_0 &= \{(ab); (a, b) \mid R(a, b) \in D\} \\
S_1 &= \{(bc); (b, c) \mid S(b, c) \in D\} \\
S_2 &= \{(ca); (c, a) \mid T(c, a) \in D\}.
\end{align*}
\]

Since none of the \( S_i \)'s is dominated, in each case both possible values occur, e.g., \( a \) and \( b \) both occur in the tuples of \( S_0 \) Thus as in Case 1, \( S_0, S_1, S_2 \) capture \( R, S, T \), respectively. We now partition \( \text{var}(q) \) into 7 sets as follows. The key idea is that for each assignment of \( x, y, z \) to values \( a, b, c \) in \( D \), we will make assignments according to that partition.

\[
\begin{align*}
\text{var}(S_0) - (\text{var}(S_1) \cup \text{var}(S_2)) &= (ab) \\
\text{var}(S_1) - (\text{var}(S_0) \cup \text{var}(S_2)) &= (bc) \\
\text{var}(S_2) - (\text{var}(S_0) \cup \text{var}(S_1)) &= (ca) \\
\text{var}(q^*) - (\text{var}(S_0) \cup \text{var}(S_1) \cup \text{var}(S_2)) &= (abc) \\
\text{var}(S_0) \cap \text{var}(S_1) &= a \\
\text{var}(S_0) \cap \text{var}(S_2) &= b \\
\text{var}(S_1) \cap \text{var}(S_2) &= c
\end{align*}
\]

We then define each other atom \( A_i \) in \( D' \) to be the following set of tuples, where the only difference between atoms is which of the 7 members of the partition of variables occurs in \( \text{var}(A_i) \):

\[
\{(ab); (bc); (ca); (abc); a; b; c \mid D \models q_\Delta(a, b, c)\}
\]  

(7)

By the definition of triad, there is a path from \( S_0 \) to \( S_1 \) not using any edges (variables) from \( S_2 \) i.e., none from \( \text{var}(S_2) \cup V_4 \cup V_6 \). Thus, any witness including occurrences of some of \( (ab), (b', b''c) \) must have \( b = b' = b'' \). Thus, as in Case 1, the witnesses of \( D' \models q \) are essentially identical to the witnesses of \( D \models q_\Delta \) and we have reduced \( \text{RES}(q_\Delta) \) to \( \text{RES}(q) \).

\( \square \)