

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

Cibele Freire  
Wellesley College  
Wellesley, USA  
cmatosfr@wellesley.edu

Neil Immerman  
University of Massachusetts Amherst  
Amherst, USA  
immerman@cs.umass.edu

Wolfgang Gatterbauer  
Northeastern University Boston  
Boston, USA  
gatterbauer@gmail.com

Alexandra Meliou  
University of Massachusetts Amherst  
Amherst, USA  
ameli@cs.umass.edu

## ABSTRACT

The resilience of a Boolean query on a database 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 conjunctive queries with self-joins, and, more specifically, we present a dichotomy result for the class of *single-self-join binary queries* with exactly two repeated relations occurring in the query. 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, confluences 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. 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.

## CCS CONCEPTS

• **Theory of computation** → **Database theory**; • **Information systems** → *Relational database model*.

## KEYWORDS

Resilience; Dichotomy; Self-join; Complexity

## ACM Reference Format:

Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2020. New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20)*, June 14–19, 2020, Portland, OR, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3375395.3387647>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
*PODS'20, June 14–19, 2020, Portland, OR, USA*  
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 978-1-4503-7108-7/20/06...\$15.00  
<https://doi.org/10.1145/3375395.3387647>

## 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.<sup>1</sup>

<sup>1</sup>While some prior work on related problems does allow for self-joins [2, 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.

In this paper, we give several novel results on the hardness of the resilience problem for CQs with self-joins. We show some results that hold for any CQ with self-join but later we focus on the class of *binary CQs* (those where relations are either unary or binary). We provide various complexity results for binary CQs where only one relation name can be repeated, which we denote by single-self-join (ssj). We analyze the case of ssj binary queries in general but emphasize that for the case with at most 2 instances of the repeated relation, we prove that a P versus NP-complete dichotomy exists. We further provide a unifying criterion for hardness (a “proof template”), and we conjecture that it subsumes and generalizes the criterion of *triads* from Sj-free queries, and that it provides a sufficient criterion of hardness for *any CQ*.

#### 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 *conjunctive queries 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 narrowing our target class to the class of *binary conjunctive queries* (those where relations are either unary or binary) and single-self-join queries (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, confluences, and permutations, and use them to prove a complete dichotomy between NP-complete and PTIME cases for the class of single-self-join binary conjunctive queries where exactly two atoms in a query correspond to the same relation (Section 7).
- We prove several involved results using the chains, confluences, and permutations structures in the case of single-self-join binary conjunctive queries where exactly 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).
- We provide the novel concept of *Independent Join Paths*. This general “proof template” aims to (i) provide a sufficient criterion of hardness for *any CQs*, (ii) subsume the prior hardness criterion of triads for SJ-free CQs, and (iii) provide a hint for an approach that could possibly automate the search for hardness reductions. (Section 9).

Some of our results apply to the general class of self-join CQs, while others apply to more restricted query families. We annotate our theoretical results with the symbols detailed in Table 1 to indicate the relevant assumptions.

Due to space restrictions, some of our proofs are omitted but we present proof sketches of some results. Please refer to the full version of this paper [15] to see all the complete proofs.

#### Query class

- all self-join conjunctive queries
- single-self-join (ssj) binary conjunctive queries
- ssj binary conjunctive queries with exactly 2 *R*-atoms
- ssj binary conjunctive queries with exactly 3 *R*-atoms

Table 1: Annotations specifying the relevant classes of queries.

## 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.,  $\mathbf{x} = (x_1, \dots, x_k)$ ) and use both subscripts and superscripts as indices (e.g.,  $a^1$  and  $a_1$ ). We fix a relational vocabulary  $\mathbf{R} = (R_1, \dots, R_\ell)$ , and denote  $\text{arity}(R_i)$  the arity of a relation  $R_i$ . We call *unary* and *binary* those relations with arity 1 or 2, respectively. We call “*binary queries*” those queries that contain only unary or binary relations. A database instance over  $\mathbf{R}$  is  $D = (R_1^D, \dots, R_\ell^D)$ , where each  $R_i^D$  is a finite relation. We call the elements of  $R_i^D$  tuples and write  $R_i$  instead of  $R_i^D$  when  $D$  is clear from the context. With some abuse of notation we also denote  $D$  as the set of all tuples, i.e.  $D = \bigcup_i R_i$ , where the union is understood to be a disjoint union (thus each tuple belongs to only one relation). The active domain  $\text{dom}(D)$  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.<sup>2</sup>

A *conjunctive query* (CQ) is a first-order formula  $q(\mathbf{y}) = \exists \mathbf{x} (g_1 \wedge \dots \wedge g_m)$  where the variables  $\mathbf{x} = (x_1, \dots, x_k)$  are called existential variables,  $\mathbf{y} = (y_1, \dots, y_c)$  are called the head variables (or free variables), and each atom (also called subgoal)  $g_i$  represents a relation  $g_i = R(\mathbf{z}_i)$  where  $\mathbf{z}_i \subseteq \mathbf{x} \cup \mathbf{y}$ .<sup>3</sup> 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, and a *single-self-join (ssj) CQ* is one where only one relation symbol can be repeated in a query. We write  $\text{var}(g_j)$  for the set of variables occurring in atom  $g_j$ . As usual, we abbreviate a non-Boolean query in Datalog notation by  $q(\mathbf{y}) :- g_1, \dots, g_m$  where  $q$  has head variables  $\mathbf{y}$  and  $g_1, \dots, g_m$  represents the body of the query.

Unless otherwise stated, a query in this paper denotes a *Boolean CQ*  $q$  (i.e.,  $\mathbf{y} = \emptyset$ ). We write  $D \models q$  to denote that the query  $q$  evaluates to true over the database instance  $D$ , and  $D \not\models q$  to denote that  $q$  evaluates to false. For a Boolean query  $q$ , we write  $q(\mathbf{x})$  to indicate that  $\mathbf{x}$  represents the set of all existentially quantified variables. We write  $[k]$  as short notation for the set  $\{1, \dots, k\}$ .

#### Additional notations.

We call a valuation of all existential variables that is permitted by  $D$  and that makes  $q$  true (i.e.  $D \models q[\mathbf{w}/\mathbf{x}]$ ) a *witness*  $\mathbf{w}$ .<sup>4</sup> The set of witnesses is then

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

<sup>2</sup>Notice that other work sometimes uses  $\text{dom}(D)$  as the size of the database. Our different definition has no implication on our complexity results but simplifies the discussions of our reductions.

<sup>3</sup>WLOG, we assume that  $\mathbf{z}_i$  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.

<sup>4</sup>Note 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].

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 this set of tuples as “witnesses.” For example, consider the query  $q_{\text{chain}} := R(x, y), R(y, z)$  with  $\mathbf{x} = (x, y, z)$  over the database  $D = \{t_1 : R(1, 2), t_2 : R(2, 3), t_3 : R(3, 3)\}$ . 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  $\{t_1, t_2\}$ ,  $\{t_2, t_3\}$ , and  $\{t_3\}$ .

In line with prior work [14, 32], relations may be specified as *exogenous*, meaning that tuples from these relations cannot be deleted.<sup>5</sup> We specify the atoms corresponding to exogenous relations with a superscript “ $x$ ”. The remaining atoms are *endogenous*.

**Complexity theory.** We write  $S \leq T$  to mean  $S \leq_{f_0} T$ .<sup>6</sup> We say that two problems have *equivalent* complexity ( $S \equiv T$ ) iff they are inter-reducible, i.e.,  $S \leq T$  and  $T \leq S$ .

## 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  $D \models q$ , what is the minimum number of endogenous tuples that have to be removed from  $D$  to make the query false? A large minimum set 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 1 (RESILIENCE DECISION).** *Given a query  $q$ , database  $D$ , and an integer  $k$ . We say that  $(D, k) \in \text{RES}(q)$  if and only if  $D \models q$  and there exists a set  $\Gamma$  with at most  $k$  endogenous tuples s.t.  $D - \Gamma \not\models q$ . We define  $\rho(D, q)$  as the size of a minimum contingency set for input  $D$  and  $q$ .*

In other words,  $(D, k) \in \text{RES}(q)$  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  $\Gamma$  as a “contingency set.” Observe that, for a fixed  $q$ , we can talk about data complexity and  $\text{RES}(q) \in \text{NP}$  when  $q$  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*  $\mathcal{H}(q)$ . 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,  $\mathcal{H}(q)$ , has vertex set  $V = \{g_1, \dots, g_m\}$ , and each variable  $x_i \in \text{var}(q)$  determines the hyperedge consisting of all those atoms in which  $x_i$  occurs:  $e_i = \{g_j \mid x_i \in \text{var}(g_j)\}$ . A *path* in the graph is an alternating sequence of vertices and edges,  $g_1, x_1, g_2, x_2, \dots, g_{n-1}, x_{n-1}, g_n$ , 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. Since we only consider dual hypergraphs, we use the shorter term “hypergraph” from now on.

<sup>5</sup>In other words, tuples in these atoms provide context and are outside the scope of possible “interventions” in the spirit of causality [18].

<sup>6</sup>First-order reductions are not required, but it is the case that all reductions defined in the paper are expressible in first-order.

**EXAMPLE 2 (HYPERGRAPHS).** *We illustrate the prior results with the following 4 queries and their hypergraphs shown in Fig. 1:*

$$q_{\Delta} := R(x, y), S(y, z), T(z, x) \quad (\text{Triangle})$$

$$q_{\text{T}} := A(x), B(y), C(z), W(x, y, z) \quad (\text{Tripod})$$

$$q_{\text{rats}} := R(x, y), A(x), T(z, x), S(y, z) \quad (\text{Rats})$$

$$q_{\text{lin}} := A(x), R(x, y, z), S(y, z) \quad (\text{Example linear})$$

In the remainder of this section, we summarize the intuition behind three main constructs—*triads*, *domination*, and *linear queries*—that lead to the result presented in Theorem 7. Then, in Section 3 we provide an exposition of how self-joins alter or completely invalidate these prior constructs.

## 2.2 Domination

We may mark some relations in an input database as exogenous and, the remaining relations are endogenous. However, some relations are “implicitly” exogenous. For example, the relation  $W$  in  $q_{\text{T}}$  is given as endogenous, but is never needed in minimum contingency sets. We next define a syntactic property, called *domination*, that captures when endogenous relations are implicitly exogenous.

**DEFINITION 3 (DOMINATION).** *If a query  $q$  has endogenous atoms  $A, B$  such that  $\text{var}(A) \subset \text{var}(B)$ , we say that  $A$  dominates  $B$ .*

For example,  $A(x)$  dominates  $W(x, y, z)$  in  $q_{\text{T}}$ . Whenever a contingency set contains tuples from  $W$ , they can always be replaced with a smaller than, or equal, number of tuples from  $A$ .

**PROPOSITION 4 (DOMINATION FOR RESILIENCE [32]).** *Let  $q$  be an sj-free CQ and  $q'$  the query resulting from labeling some dominated atoms as exogenous. Then  $\text{RES}(q) \equiv \text{RES}(q')$ .*

When studying resilience, we follow the convention that *all dominated atoms are made exogenous*, and we consider that the normal form of a query. As we have seen,  $A$  dominates  $W$  in  $q_{\text{T}}$ . Similarly, the atom  $A$  dominates both  $R$  and  $T$  in  $q_{\text{rats}}$ . We thus transform the queries so that the dominated atoms are exogenous. Exogenous atoms have the superscript “ $x$ ”.

$$q'_{\text{T}} := A(x), B(y), C(z), W^x(x, y, z)$$

$$q'_{\text{rats}} := R^x(x, y), A(x), T^x(z, x), S(y, z)$$

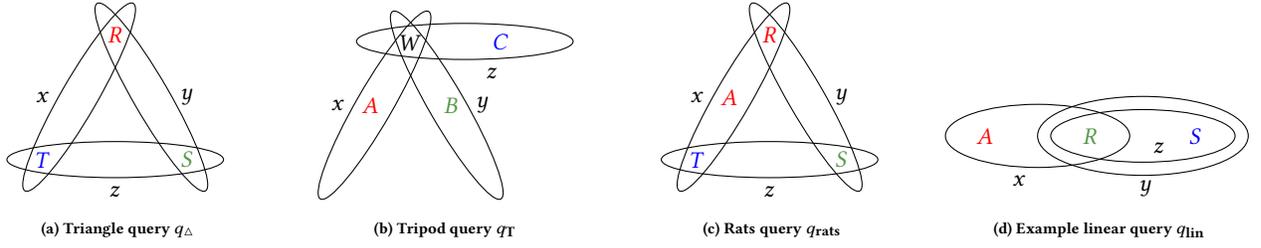
Proposition 4 implies that  $\text{RES}(q_{\text{rats}}) \equiv \text{RES}(q'_{\text{rats}})$ .

## 2.3 Triads and hardness

We showed in [14] that  $\text{RES}(q_{\Delta})$  and  $\text{RES}(q_{\text{T}})$  from Example 2 are NP-complete. While  $q_{\Delta}$  and  $q_{\text{T}}$  appear to be quite different, they share a key common structural property which alone is responsible for hardness for sj-free CQs.

**DEFINITION 5 (TRIAD).** *A triad is a set of three endogenous atoms,  $\mathcal{T} = \{S_0, S_1, S_2\}$  such that for every pair  $i, j$ , there is a path from  $S_i$  to  $S_j$  in  $\mathcal{H}(q)$  that uses no variable occurring in the other atom of  $\mathcal{T}$ .*

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  $q_{\text{T}}$  (see Fig. 1). 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:



**Figure 1: The hypergraphs of queries  $q_\Delta$ ,  $q_T$  and  $q_{\text{lin}}$ .**  $\{R, S, T\}$  is a triad of  $q_\Delta$  and  $\{A, B, C\}$  is a triad of  $q_T$ . Thus  $\text{RES}(q_\Delta)$  and  $\text{RES}(q_T)$  are NP-complete. In contrast,  $A$  dominates both  $R$  and  $T$  in  $q_{\text{rats}}$  which renders both atoms exogenous, thus “disarming” what appears to be a triad. And  $\mathcal{H}(q_{\text{lin}})$  is linear. Thus  $\text{RES}(q_{\text{rats}})$  and  $\text{RES}(q_{\text{lin}})$  are in P.

LEMMA 6 (TRIADS MAKE  $\text{RES}(q)$  HARD [14]). *Let  $q$  be an sj-free CQ where all “dominated” atoms are exogenous. If  $q$  has a triad, then  $\text{RES}(q)$  is NP-complete.*

## 2.4 Linear queries

A query  $q$  is *linear* if its atoms can be arranged in a linear order s.t. each variable occurs in a contiguous sequence of atoms. Geometrically, a query is linear if all of the vertices of its hypergraph can be drawn along a straight line and all of its hyperedges can be drawn as convex regions (thus the variables form intervals on a line of relations). For example  $q_{\text{lin}}$  is linear (see Fig. 1d).

It was shown in [32] that for any sj-free CQ that is linear,  $\text{RES}(q)$  may be computed in a natural way using network flow. Thus all such queries are easy.

If all sj-free CQs without a triad were linear, then this would complete the dichotomy theorem for resilience. While this is not the case, we completed the proof of Theorem 7, by showing that *every triad-free sj-free CQ may be transformed to a linear query of equivalent resilience.*

## 2.5 Dichotomy Theorem

Now we can present the full characterization of the complexity of sj-free CQs proved in [14].

THEOREM 7 (DICHOTOMY OF RESILIENCE FOR SJ-FREE CQs [14]). *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  $\text{RES}(q)$  is NP-complete, otherwise it is in PTIME.*

## 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 lead to the desired properties in the presence of self-joins. Section 3.2 explains why dominated atoms may still be relevant when computing the minimum contingency set. (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., “ $R(x, y), R(y, y)$ .”

In the cases where the variable position is relevant and we are restricted to binary queries, we naturally represent queries as labeled direct graphs. This representation captures all relevant structural information of the binary queries, especially the relative position of variables, which the hypergraph representation does not reflect.

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

### 3.1 Basic hard queries: $q_{\text{vc}}$ and $q_{\text{chain}}$

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

$$\begin{aligned} q_{\text{vc}} &:- R(x), S(x, y), R(y) && \text{(Vertex cover)} \\ q_{\text{chain}} &:- R(x, y), R(y, z) && \text{(Chain query)} \end{aligned}$$

Figure 2 shows graphical representations of both queries while illustrating the differences between the *dual hypergraph* and the *binary graph* of a binary CQ.

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

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

### 3.2 SJ-Free domination no longer works

We saw from Proposition 4 that in sj-free CQs, making all dominated atoms exogenous leaves the query resilience unchanged. In the presence of self-joins, this is no longer true.

EXAMPLE 11. *Query  $q_{\text{rats}}^{\text{sj}_1} := A(x), R(x, y), R(y, z), R(z, x)$  is a self-join variation of  $q_{\text{rats}}$  with  $S, T$  replaced by  $R$ ’s. Similar to  $q_{\text{rats}}$ , we have  $\text{var}(A) \subseteq \text{var}(R(x, y))$ , so  $A$  dominates  $R$  by Definition 3. Thus  $R$  should become exogenous when searching for the minimal contingency set. But this is not the case. Consider the database instance*

$$D = \{A(1), A(5), R(1, 2), R(2, 3), R(3, 1), R(5, 1), R(2, 5)\}$$



Figure 2: *Hypergraphs* only represent which variables occur in a given atom, whereas *binary graphs* represent containment and position within each atom. Both concepts are illustrated here for two basic hard CQs with self-joins:  $q_{\text{chain}}$  and  $q_{\text{vc}}$ .

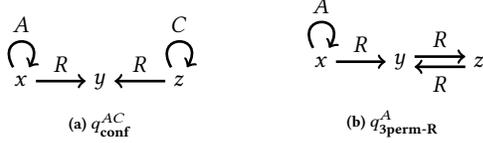


Figure 3: Two example of PTIME queries that require modified version of network flow. Notice that  $q_{3\text{-perm-R}}^A$  contains the hard query  $q_{\text{chain}}$  and is still in P.

Our query has 3 witnesses over this database:  $(1, 2, 3)$ ,  $(1, 2, 5)$ , and  $(5, 1, 2)$ . If  $R$  was made exogenous, the only possible minimum contingency set would be  $\Gamma = \{A(1), A(5)\}$ . However, if  $R$  is considered as endogenous, there is a smaller contingency set, with only  $R(1, 2)$ .

This example shows that domination as defined in Definition 3 no longer implies that a relation can be made exogenous in the self-join case. This immediately raises the question of whether there is a set of conditions which implies that a relation can be made exogenous in the self-join setting, i.e. if there is a self-join version of domination. Additionally, does  $q_{\text{rats}}^{\text{sj}_1}$  have a triad? The answer to both is yes, as we will see in Section 4.3 and Section 5.1, respectively.

### 3.3 Easy queries that use flow in a trickier way

As mentioned in the discussion of Theorem 7, 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.

The following queries are two such examples, where modified versions of network flow are used to show resilience is easy in these cases.

$$q_{\text{conf}}^{\text{AC}} := A(x), R(x, y), R(z, y), C(z)$$

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

PROPOSITION 12 ( $q_{\text{CONF}}^{\text{AC}}$ ).  $\text{RES}(q_{\text{CONF}}^{\text{AC}})$  is in P.

PROPOSITION 13 ( $q_{3\text{perm-R}}^A$ ).  $\text{RES}(q_{3\text{perm-R}}^A)$  is in P.

## 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.

### 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.

### 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,

$$q_{\text{comp}} := A(x), R(x, y), R(z, w), B(w)$$

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 resiliences 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  $(q, D)$ .

LEMMA 14 (● QUERY COMPONENTS). Let  $q := q_1, \dots, q_k$  be a query that consists of  $k$  components  $q_i$ ,  $i \in [k]$ . Then  $\rho(q, D) = \min_i \rho(q_i, D)$ .

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

LEMMA 15 (● QUERY COMPONENTS COMPLEXITY). Let  $q$  be a minimal query that consists of  $k$  query components.  $\text{RES}(q)$  is NP-complete if there is at least one component  $i \in [k]$  for which  $\text{RES}(q_i)$  is NP-complete. Conversely, if  $\text{RES}(q_i)$  is in P for all  $i$ , then  $\text{RES}(q)$  is in P.

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

### 4.3 SJ-domination

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  $\text{pos}_g^q(i) = x$  to express that the  $i$ -th attribute of atom  $g$  is variable  $x$  for a query  $q$  and omit  $q$  when  $q$  is clear from the context.

DEFINITION 16 (DOMINATION WITH SELF-JOINS). *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  $B$  atom  $g_B$ , there exists an  $A$  atom  $h_A$  satisfying  $\text{pos}_{h_A}(i) = \text{pos}_{g_B}(f(i))$ ,  $\forall i \in [\text{arity}(A)]$ . In other words, each  $B$  atom that occurs in  $q$  has a corresponding  $A$  atom, and each of these pairs will have matching variables accordingly to function  $f$ .*

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

EXAMPLE 17. *To illustrate the new self-join domination, consider the following queries:*

$$\begin{aligned} q_1 &:- R(x, y), A(y), R(y, z), S(y, z) \\ q_2 &:- R(x, y), A(y), R(z, y), S(y, z) \end{aligned}$$

*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  $A(b)$ .*

PROPOSITION 18 (DOMINATION FOR RESILIENCE WITH SELF-JOIN). *Let  $q$  be a CQ and  $q'$  the result of labeling some dominated relations exogenous. Then  $\text{RES}(q) \equiv \text{RES}(q')$ .*

#### 4.4 Outline of our plan of attack

To obtain a dichotomy result for the resilience of binary queries in the presence of a single-self-join, we proceed as follows. (1) Section 5 shows that triads in any conjunctive queries with self-joins still imply hardness (Theorem 24) and furthermore, when triads are absent, the endogenous atoms are linearly connected. We call such queries *pseudo-linear* (Theorem 25). We conjecture that pseudo-linear queries may be transformed to linear queries of equivalent resilience (Conjecture 26). In any case, it suffices to study the criteria for hardness of pseudo-linear queries. (2) Section 6 generalizes the hardness pattern behind  $q_{vc}$  to a more general class of hard ssj binary queries that contain “paths” between repeated atoms. (3) We then focus on the complexity of the resilience of ssj 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 relation. This is a dichotomy theorem: we show that for all such queries,  $q$ ,  $\text{RES}(q)$  is either NP-complete or  $\text{RES}(q)$  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 3 occurrences of a repeated relation. In Section 9, we present a “template” for hardness proofs that we believe will help us make progress in the general self-join case.

### 5 NON-LINEAR QUERIES: NP-COMPLETE

In this section we prove that queries containing triads remain hard in the presence of self-joins (Theorem 24). 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.

DEFINITION 19 (SELF-JOIN VARIATION OF A CQ). *Let  $q$  be a sj-free CQ and let  $q^{sj}$  result from  $q$  by replacing some atoms  $S_i(\bar{v})$  from  $q$  with the atom  $R_i(\bar{v})$ , where the relation  $R_i$  occurs elsewhere in  $q$ . We say that  $q^{sj}$  is a self-join variation of  $q$ .*

EXAMPLE 20 (SELF-JOIN VARIATIONS). *Consider sj-free query  $q_\Delta$ . The following are all its possible self-join variations:*

$$\begin{aligned} q_\Delta &:- R(x, y), S(y, z), T(z, x) && \text{(Triangle)} \\ q_\Delta^{sj_1} &:- R(x, y), R(y, z), R(z, x) \\ q_\Delta^{sj_2} &:- R(x, y), R(y, z), T(z, x) \\ q_\Delta^{sj_3} &:- R(x, y), S(y, z), R(z, x) \end{aligned}$$

We first observe that the resilience of self-join variations of a query can only be harder than their sj-free counterpart:

LEMMA 21 (SJ CAN ONLY MAKE RESILIENCE HARDER). *Let  $q$  be an sj-free CQ and let  $q^{sj}$  be a self-join variation of  $q$ . If  $q^{sj}$  is minimal, then  $\text{RES}(q) \leq \text{RES}(q^{sj})$ .*

We need to rely on the fact that  $q^{sj}$  is minimal for the result to hold, as we see in the example below:

EXAMPLE 22. *Consider query  $q :- R(x, y), S(z, y), T(z, w), A(x, w)$ , and observe that  $\text{RES}(q)$  is NP-complete because  $q$  contains a triad. A possible self-join variation is  $q^{sj} :- R(x, y), R(z, y), R(z, w), R(x, w)$ . Note that  $q^{sj}$  is not minimal, and is equivalent to  $R(x, y)$ . So  $\text{RES}(q^{sj})$  is trivially in P.*

By Lemma 21, the resilience of the self-join variations of  $q_\Delta$  are NP-complete. Recall from Definition 5 that a triad is set of three endogenous atoms, so we can say that the self-join variations of  $q_\Delta$  all have triads. However, it does not immediately follow from Lemma 21 that every sj-query with a triad is hard. The missing cases are when an sj-query includes a triad, but it is not a self-join variation of an sj-free query with a triad. We next explore this situation.

#### 5.1 Self-join variations of $q_{\text{rats}}$ and $q_{\text{brats}}$

Recall two important sj-free queries:

$$\begin{aligned} q_{\text{rats}} &:- R(x, y), A(x), T(z, x), S(y, z) && \text{(Rats)} \\ q_{\text{brats}} &:- B(y), R(x, y), A(x), T(z, x), S(y, z) && \text{(Brats)} \end{aligned}$$

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

The same doesn't occur with some self-join variations of  $q_{\text{rats}}$  and  $q_{\text{brats}}$ . Below we list two example of variations which contain triads.

$$\begin{aligned} q_{\text{rats}}^{sj_1} &:- R(x, y), A(x), R(y, z), R(z, x) \\ q_{\text{brats}}^{sj_1} &:- B(y), R(x, y), A(x), R(z, x), R(y, z) \end{aligned}$$

In these examples, relation  $R$  is now more robust and not dominated by  $A$  or  $B$ . Therefore, they still contain triads consisting of their three  $R$ -atoms. The presence of a triad is a strong indication

that these queries are hard but we cannot use Lemma 21 to show this because their sj-free counterparts,  $q_{\text{rats}}$  and  $q_{\text{brats}}$ , are easy. We now proceed to show their complexity is hard.

**PROPOSITION 23.** *Let  $q$  be a self-join variation of  $q_{\text{rats}}$  or  $q_{\text{brats}}$ . If  $q$  has a triad, then  $\text{RES}(q)$  is NP-complete.*

Using Proposition 23, we now generalize the fact that triads make sj-free queries hard (Lemma 6) to the same result for general CQs.

## 5.2 Triads Make Queries Hard

**THEOREM 24** (● SJ-QUERIES WITH TRIADS). *If  $q$  has a triad, then  $\text{RES}(q)$  is NP-complete.*

**PROOF SKETCH.** We argue that there are only two cases to consider when a sj-query  $q$  has a triad. Consider that  $q$  is a self-join variations of a sj-free query  $q'$ . The first case is when the resilience of  $q'$  is hard, i.e.,  $\text{RES}(q')$  is NP-complete. Then, we can use Lemma 21 to show  $\text{RES}(q)$  is NP-complete. The second case is when  $\text{RES}(q')$  is in P. Then, we argue that we can show a reduction from  $\text{RES}(q'') \leq \text{RES}(q)$ , where  $q''$  is a self-join variation of either  $q_{\text{rats}}$  or  $q_{\text{brats}}$  that contains a triad. From Proposition 23,  $\text{RES}(q'')$  is NP-complete, which thus implies that  $\text{RES}(q)$  is also NP-complete.  $\square$

Thus, if a query contains a triad, it is hard. In the next section, we discuss queries that do not contain triads and how they are similar to linear queries, since their endogenous atoms 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  $\text{RES}(q) \leq \text{RES}(q')$ . 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,

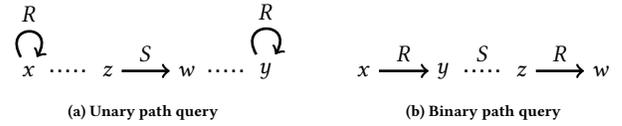
**THEOREM 25** (● NO TRIAD MEANS PSEUDO-LINEAR). *Let  $q$  be a CQ with no triad. Then all endogenous atoms in  $q$  are connected linearly.*

We conjecture that pseudo-linearity is equivalent to linearity when considering resilience. What makes a query pseudo-linear, instead of linear or containing a triad, is the presence of some exogenous atoms. However, the exogenous atoms of a query are mostly only connecting the endogenous atoms, and also, if necessary, ensuring that  $q$  is a minimal query, so we believe they can be modified to obtain a linear query without altering the complexity of a query.

**CONJECTURE 26** (● 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. Note that these queries



**Figure 4: General structure of path queries.**

are binary and, in both, only one relation is part of a self-join. In other words, these are single-self-join binary queries.

We now identify a pattern characteristic of  $q_{\text{vc}}$  that we call a *path*. The main result of this section is that every ssj binary query containing a path is hard. We start by showing the case where the self-join relation is unary.

**THEOREM 27** (● UNARY PATH). *Let  $q$  be a minimal ssj-CQ. If  $q$  contains distinct atoms  $R(x)$  and  $R(y)$ , then  $\text{RES}(q)$  is NP-complete.*

**PROOF SKETCH.** Let  $R(x)$  and  $R(y)$  be the first two occurrences of the relation  $R$  in  $q$ . Since  $q$  is connected,  $R(x)$  and  $R(y)$  are connected by at least one non-self-join relation,  $S$  (see Fig. 4a). We prove that  $\text{RES}(q_{\text{vc}}) \leq \text{RES}(q)$ . The full proof is in [15], 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  $R', S'$  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,  $R$ , to the right of  $R(y)$ ) 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.  $\square$

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

**THEOREM 28** (● BINARY PATH). *Let  $q$  be a minimal ssj-CQ. If  $q$  has distinct consecutive sj atoms  $R(x, y)$ ,  $R(z, w)$  with  $\{x, y\} \cap \{z, w\} = \emptyset$ , then  $\text{RES}(q)$  is NP complete.*

**PROOF SKETCH.** Given  $R(x, y)$ ,  $R(z, w)$  as in the statement of the theorem, there must be an atom  $S(u, v)$ , with  $S \neq R$  on the path between them, and  $u \in \{x, y\}$  and  $v \notin \{x, y\}$ . Now, as in the proof of Theorem 27, we reduce  $\text{RES}(q_{\text{vc}})$  to  $\text{RES}(q)$ . We map any database  $D \models q_{\text{vc}}$  to a database  $D' \models q$ , where  $R'$  contains  $\{(a, a) \mid A(a) \in D\}$  plus other multiple, extra values for any other atoms of the relation  $R$  in  $q$  to the left of  $R(x, y)$  or the right of  $R(z, w)$  and  $S' = \{(a, b) \mid R(a, b) \in D\}$ . Same as in the unary case, there is no assumption about the linearity of the query.  $\square$

Unary and Binary Paths are the simplest of the hard patterns. By Theorem 27 and Theorem 28, they always force their queries to be hard.

Since we have established that an sj- query either has a triad or is pseudo-linear (Theorem 25) and because we have proved that triads imply hardness (Theorem 24), we can now focus on the pseudo-linear queries.

In the next sections we study the more subtle pseudo-linear ssj binary queries, which do not contain paths.

## 7 QUERIES WITH EXACTLY TWO R-ATOMS

In this section we cover the complexity of pseudo-linear ssj binary queries with exactly two atoms referring to the same relation. We will refer to this relation as  $R$ . 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 27 and Theorem 28; otherwise we would already know that  $\text{RES}(q)$  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,  $R$  must be a binary relation and the two  $R$ -atoms 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.,  $R(x, y), R(z, y)$ ;
- *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  $R$ -atom e.g.,  $R(x, x), R(x, y), B(y)$

Figure 5 shows the binary graphs for each these patterns, which helps visualize the subtle variations in how the  $R$ -atoms can join. 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 expansions remains NP-complete.

We call “expansions” of  $q_{\text{chain}}$  any query obtained by adding new relations to it, i.e. relations that do not self-join. We start by presenting the expansions obtained by adding unary relations and then generalize that to any expansion.

Figure 6a shows how unary relations can be added to  $q_{\text{chain}}$ . 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 expansions are hard.

**PROPOSITION 29 (CHAINS WITH UNARY RELATIONS).** *Any expansion of  $q_{\text{chain}}$  with unary relations is NP-complete.*

**PROOF SKETCH.** We prove these expansions are hard by a reduction from 3SAT. The same idea used to prove that  $\text{RES}(q_{\text{chain}})$  is hard will work here as long as we adapt the variable and clause gadgets to deal with the existence of the unary relations.  $\square$

Now we can generalize this hardness result to any chain expansion using a reduction idea similar to the ones used for the proofs of Theorems 27 and 28 for paths.

**PROPOSITION 30 (•: CHAINS).** *If a query  $q$  contains a 2-chain as its only self-join, then  $\text{RES}(q)$  is NP-complete.*

### 7.2 2-Confluences

*Confluences* are defined by relation  $R$  joining only in the same attribute. We refer to this pattern as  $q_{\text{conf}}$  (Fig. 6b).

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

We next show that the standard flow 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 31 (•:  $q_{\text{CONF}}$ ).**  *$\text{RES}(q)$  for any linear query  $q$  with  $q_{\text{conf}}$  as its only self-join pattern can be solved in P by standard network flow.*

In Proposition 31 we assume that  $q$  is linear, thus guaranteeing that every path in  $q$  from  $x$  to  $z$  involves the variable  $y$ , and therefore we are able to create a network flow to solve the problem. Note that this is not true in general for pseudo-linear queries containing  $q_{\text{conf}}$ . For example, consider  $cf_p := R(x, y)H^x(x, z)R(z, y)$ . It is easy to see that  $cf_p$  is pseudo-linear but we have  $\text{RES}(cf_p) \equiv \text{RES}(q_{\text{vc}})$ . Thus, we cover all possible cases for  $q_{\text{conf}}$  by observing,

**PROPOSITION 32 (•:).** *Let  $q$  be a pseudo-linear query with  $q_{\text{conf}}$  as its only self-join pattern. 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 P.*

### 7.3 2-Permutations

We call two  $R$ -atoms sharing both variables a *permutation*. The smallest pattern that has this property is  $R(x, y), R(y, x)$  (Fig. 5). We show that permutations have both NP-complete and PTIME instances.

**Easy permutations.** We start with two easy permutations.

$$q_{\text{perm}} := R(x, y), R(y, x) \quad q_{\text{perm}}^A := A(x), R(x, y), R(y, x)$$

**PROPOSITION 33.**  *$\text{RES}(q_{\text{perm}})$  and  $\text{RES}(q_{\text{perm}}^A)$  are in P.*

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

Given a database  $D_2$  satisfying  $q_{\text{perm}}^A$ , for each join  $(a, b)$ , we have 2 possible choices. Either  $A(a)$  will be in the min  $\Gamma$  or either one of  $R(a, b)$  and  $R(b, a)$  but never both. Therefore we can reduce  $\text{RES}(q_{\text{perm}}^A)$  to vertex cover in a bipartite graph, which is in P.  $\square$

**Hard permutations.** Surprisingly, adding another unary atom to  $q_{\text{perm}}^A$ , thus bounding it on both ends, leads to a hard query.

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

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

**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

| Pattern Name       | Binary Graph                            | PTIME cases   | NP-hard cases   |
|--------------------|---|---------------|-----------------|
| $q_{\text{chain}}$ | $x \xrightarrow{R} y \xrightarrow{R} z$ | No PTIME case |                 |
| $q_{\text{conf}}$  | $x \xrightarrow{R} y \xleftarrow{R} z$  |               |                 |
| $q_{\text{perm}}$  | $x \xleftrightarrow{R} y$               |               |                 |
| REP                |   |               | No NP-hard case |

Figure 5: Binary graphs representing all the possible self-join patterns with two  $R$ -atoms, where  $R$ -atoms share at least one variable.

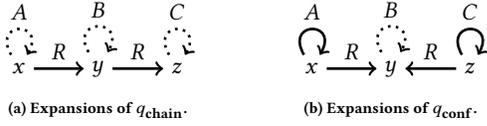


Figure 6: Expansions of  $q_{\text{chain}}$  and  $q_{\text{conf}}$  with unary relations.

hard permutation,  $q_{\text{perm}}^{AB}$ , is bound, but the easy ones,  $q_{\text{perm}}$ ,  $q_{\text{perm}}^A$ , are not bound. Using this characterization, we identify when 2-permutations are hard.

PROPOSITION 35 (●). *Let  $q$  be a pseudo-linear 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 atoms contain the same variable twice, e.g. occurrences of  $R(x, x)$ . Note that this is only relevant for the case where  $R$  is part of a self-join, otherwise it could be considered as  $R(x)$ .

There are only three patterns to consider when we are restricted to two  $R$ -atoms, either one or both atoms have repeated variables. The following queries are the smallest examples of this class of queries:

$$\begin{aligned} z_1 &:- R(x, x), S(x, y), R(y, y) \\ z_2 &:- R(x, x), S(x, y), R(y, z) \\ z_3 &:- R(x, x), R(x, y), A(y) \end{aligned}$$

Notice that queries  $z_1$  and  $z_2$  satisfy the condition for hardness of binary paths (Theorem 28), since their set of variables is disjoint. Therefore, we can conclude that  $\text{RES}(z_1)$  and  $\text{RES}(z_2)$  are NP-complete, as well as any expansion of those queries. We show that any REP queries that contain  $z_3$  are in P.

PROPOSITION 36 (●). *Any pseudo-linear query  $q$  with exactly two  $R$ -atoms that contains  $z_3$  is in P.*

## 7.5 The dichotomy

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

THEOREM 37 (●) TWO-ATOM DICHOTOMY. *Consider  $q$  an ssj-CQ, with at most two occurrences of the self-join relation. If  $q$  has any of the following*

- (1) *triad*
- (2) *path*
- (3) *chain*
- (4) *bounded permutation*
- (5) *confluence with exogenous path*

*then  $\text{RES}(q)$  is NP-complete. Otherwise,  $\text{RES}(q)$  is PTIME via a reduction to network flow. In addition there is a PTIME algorithm that on input  $q$  determines which case occurs.*

## 8 QUERIES WITH EXACTLY THREE R-ATOMS

In Theorem 37 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.

$$q_{3\text{chain}} := R(x, y), R(y, z), R(z, w)$$

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

PROPOSITION 38 (●). 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.

$$q_{3\text{conf}} := R(x, y), R(z, y), R(z, w)$$

As in the 2-confluence case,  $q_{3\text{conf}}$  is not minimal, so other atoms are required to make it minimal. Here are a few examples of minimal queries containing  $q_{3\text{conf}}$ .

$$q_{3\text{conf}}^{AC} := A(x), R(x, y), R(z, y), R(z, w), C(w)$$

$$q_{3\text{conf}}^{TS} := T(x, y)^x, R(x, y), R(z, y), R(z, w), S(z, w)^x$$

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

PROPOSITION 39.  $\text{RES}(q_{3\text{conf}}^{AC})$  is NP-complete.

PROPOSITION 40 (●). Any variation of  $q_{3\text{conf}}^{AC}$  obtained by including unary relations is NP-complete.

PROOF. We define a reduction from Max 2SAT similar to the one used for  $q_{3\text{conf}}^{AC}$  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 41.  $\text{RES}(q_{3\text{conf}}^{TS})$  is in P.

**Open problem.** There is a third variant of 3-confluences which somewhat mix queries  $q_{3\text{conf}}^{AC}$  and  $q_{3\text{conf}}^{TS}$  (Fig. 7).

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

The complexity of  $\text{RES}(q_{3\text{conf}}^{AS})$  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.

$$q_{3\text{cc}}^{AC} := A(x)R(x, y)R(y, z)R(w, z)C(w)$$

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

$$q_{3\text{cc}}^C := R(x, y)R(y, z)R(w, z)C(w)$$

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 42.  $\text{RES}(q_{3\text{cc}}^{AC})$  and  $\text{RES}(q_{3\text{cc}}^{AS})$  are NP-complete.

PROPOSITION 43.  $\text{RES}(q_{3\text{cc}}^C)$  is NP-complete.

**Open Problem.** In this category of queries with chain and confluence, we don't know the complexity of  $q_{3\text{cc}}^S := R(x, y)R(y, z)R(w, z)S(w, z)$ .

## 8.4 3-Permutation plus R

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

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

Similar to the  $q_{3\text{conf}}$  case,  $q_{3\text{perm-R}}$  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. 3b).

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

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

We proved in Proposition 13 that  $\text{RES}(q_{3\text{perm-R}}^A)$  is in P by using network flow. A similar argument proves that  $\text{RES}(q_{3\text{perm-R}}^{S_{wx}})$  is also in P.

PROPOSITION 44.  $\text{RES}(q_{3\text{perm-R}}^{S_{wx}})$  is in P.

The next query we will see is  $q_{3\text{perm-R}}^{S_{xy}}$ . Although very similar to  $q_{3\text{perm-R}}^A$  and  $q_{3\text{perm-R}}^{S_{wx}}$ ,  $\text{RES}(q_{3\text{perm-R}}^{S_{xy}})$  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 34.

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

PROPOSITION 45.  $\text{RES}(q_{3\text{perm-R}}^{S_{xy}})$  is NP-complete.

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

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

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

$$q_{3\text{perm-R}}^{S_{xy}BC} := S(x, y)R(x, y)B(y)R(y, z)R(z, y)C(z)$$

PROPOSITION 46.  $\text{RES}(q_{3\text{perm-R}}^{AC})$ ,  $\text{RES}(q_{3\text{perm-R}}^{AB})$  and  $\text{RES}(q_{3\text{perm-R}}^{S_{xy}BC})$  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:

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

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

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

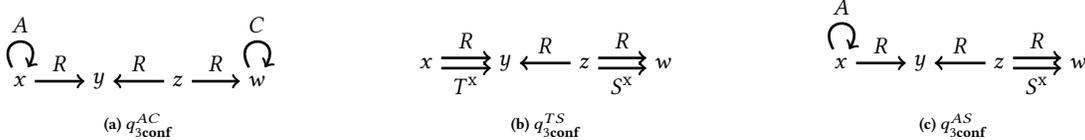


Figure 7: Three main queries containing a 3-confluence.

## 8.5 Queries with REP

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

$$z_4 := R(x, x)R(x, y)S(x, y)R(y, y)$$

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

PROPOSITION 47.  $\text{RES}(z_4)$  and  $\text{RES}(z_5)$  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.

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

$$z_7 := A(x)R(x, y)R(y, x)R(y, y)$$

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

## 9 INDEPENDENT JOIN PATHS: A UNIFYING HARDNESS CRITERION

**Motivation.** We now define a particular “template” for hardness reductions which we call *Independent Join Paths* or IJPs. The idea is that if we can construct a particular database that fulfills 5 criteria for a query  $q$ , then we can conclude safely that  $\text{RES}(q)$  is NP-complete.

This recent development is exciting for several reasons: 1) In our earlier attempts to prove hardness for queries, we amassed a plethora of different hardness proofs, with little immediate intuition of how one hardness proof immediately helps facilitate the hardness proof of another query. Now we expect that the task can be simplified to the task of searching for any particular database that serves as “proof” of hardness based on a generalized reduction from Vertex Cover. 2) We were able to look at our existing hardness proofs and post-hoc identify some part in some gadget that formed an IJP. In other words, *IJPs were already present in our hardness proofs* (we give examples in Appendix A). Thus IJPs are really a *unifying* common denominator for all hard queries known so far. 3) The search for hardness proofs could now, in theory, be automated. While we have not yet explored this idea, we give the intuition in Appendix A. 4) The hardness based on IJPs is not restricted to the particular fragment of CQs that we have analyzed in this paper; rather they are a universal criterion. Even the original criterion of *triads* for sj-free CQs can be subsumed under IJPs. 5) We conjecture that the inability to form IJPs for those queries that are in PTIME can be deduced from the structure of a query, and future work will discover the reason.

**The intuition of IJPs.** We have already seen that paths between two subgoals  $g_1$  and  $g_n$  that refer to the same relation are a sufficient condition for hardness under “certain circumstances”. Recall our

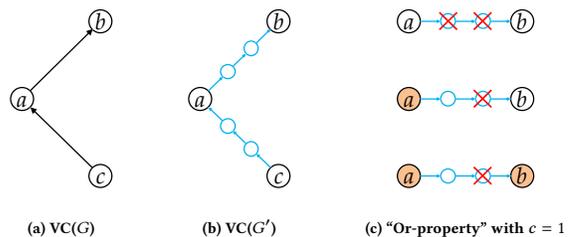


Figure 8: Intuition behind IJPs

simplest example for a path implying hardness:  $q_{\text{vc}} := R(x), S(x, y), R(y)$ . The intuition of our construction is now as follows: Take any minimal VC problem for a graph  $G(V, E)$  (see Fig. 8a). Replace any existing arc with 3 arcs instead to create  $G'$  (see Fig. 8b). Then  $G$  has a VC of size  $k$  iff  $G'$  has a VC of size  $k + |E|$ . Similarly, replace each arc with 5 instead of 3 arcs, then the condition for  $G'$  is  $k + 2|E|$ . The key property we needed for this to work is the fact that 3 arcs form a particular path with the following “OR-property” (see Fig. 8c): As long as at least one end point of the path is removed, then the minimal VC is exactly one additional node per path.

**Formalization of IJPs.** We next use this idea to define a particular canonical database instance which we call “Independent Join Path.” We conjecture that whenever a query has such a canonical database, then resilience is hard by a proof that generalizes the idea from above. We give the formal definition here and provide intuition for each of the conditions in Appendix A. In the following, we write  $\mathbf{x}_j$  to denote the subvector of  $\mathbf{x}$  that retains only the entries indexed by  $j$ . For example if  $\mathbf{x} = (1, 2, 3, 4, 5)$  and  $j = (2, 4, 5)$  then  $\mathbf{x}_j = (2, 4, 5)$

DEFINITION 48 (INDEPENDENT JOIN PATH). *A database  $D$  forms an Independent Join Path for query  $q$  if the following conditions hold:*

- (1) *There is a relation  $R$  containing at least two tuples  $R(\mathbf{a})$  and  $R(\mathbf{b})$  with  $\mathbf{a} \not\subseteq \mathbf{b}$  and  $\mathbf{b} \not\subseteq \mathbf{a}$ .*
- (2) *In  $D$ ,  $R(\mathbf{a})$  and  $R(\mathbf{b})$  each participate in exactly one witness  $\mathbf{w}_a, \mathbf{w}_b$  of  $D \models q$ . Both  $\mathbf{w}_a$  and  $\mathbf{w}_b$  have exactly  $m$  tuples, where  $m$  is the number of atoms in  $q$ .*
- (3) *There is no endogenous relation  $S$  containing a tuple  $S(\mathbf{c})$  with  $\mathbf{c} \subset \mathbf{a}$  or  $\mathbf{c} \subset \mathbf{b}$ .*
- (4) *If there is an exogenous relation  $T^x$  containing a tuple  $T^x(\mathbf{d})$  with  $\mathbf{d} = \mathbf{a}_j$  for some  $j$ , then  $T^x$  also contains  $T^x(\mathbf{e})$  with  $\mathbf{e} = \mathbf{b}_j$ .*
- (5) *Let  $c$  be the resilience of  $q$  on  $D$ :  $\rho(q, D) = c$ . Then the resilience is  $c - 1$  in all 3 cases of removing either  $R(\mathbf{a})$ , or  $R(\mathbf{b})$ , or both.*

CONJECTURE 49 (IJPs IMPLY HARDNESS). *If there is a database  $D$  that forms an IJP for a query  $q$ , then  $\text{RES}(q)$  is NP-complete.*

**The conjecture.** For the fragment of CQs we are considering in this paper, we have been able to simplify some hardness proofs, which at times use very different constructions (reductions from

VC, 3-SAT, Max 2-SAT), by looking at our existing hardness proofs and identifying IJPs in our *existing* gadgets.

We conjecture that the existence of IJPs for a query is also a *necessary* condition for hardness, that there is an algorithm to verify whether a query can form IJPs or not, and that the fact that a query cannot form IJPs (such as linear SJ-free CQs) translates immediately into a PTIME algorithm for solving  $\text{RES}(q)$ .

## 10 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 consequentially 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, 16, 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, 19–21, 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, 17]. “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.

## 11 FINAL REMARKS

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

We also present results towards the for the case of binary CQs with a single self-join relation that appears in 3 atoms, and identifies some open problems and challenges towards completing the dichotomy for this class (Section 8).

Our work also presents a roadmap for tackling the analysis of more extended query families. Section 9 provides towards a possible generalization of our results to all class of self-join queries, by using a unifying criterion that we call *Independent Join Paths*.

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.

**Acknowledgements.** This work was supported in part by the National Science Foundation under grants IIS-1453543, CCF-1617498, IIS-1762268, and CCF-1763423.

## REFERENCES

- [1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases: The Logical Level*. Addison-Wesley.
- [2] Antoine Amarilli, Mikaël Monet, and Pierre Senellart. 2017. Conjunctive Queries on Probabilistic Graphs: Combined Complexity. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '17)*. ACM, New York, NY, USA, 217–232. <https://doi.org/10.1145/3034786.3056121>
- [3] F. Bancilhon and N. Spyratos. 1981. Update Semantics of Relational Views. *ACM TODS* 6, 4 (Dec. 1981), 557–575. <https://doi.org/10.1145/319628.319634>
- [4] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. 2001. Why and Where: A Characterization of Data Provenance. In *ICDT*. 316–330.
- [5] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. 2002. On Propagation of Deletions and Annotations Through Views. In *PODS*. 150–158. <https://doi.org/10.1145/543613.543633>
- [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/10.1145/800105.803397>
- [7] Adriane Chapman and H. V. Jagadish. 2009. Why not?. In *SIGMOD*. 523–534.
- [8] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases* 1, 4 (2009), 379–474.
- [9] Gao Cong, Wenfei Fan, Floris Geerts, Jianzhong Li, and Jizhou Luo. 2012. On the Complexity of View Update Analysis and Its Application to Annotation Propagation. *IEEE TKDE* 24, 3 (2012), 506–519. <https://doi.org/10.1109/TKDE.2011.27>
- [10] Stavros S. Cosmadakis and Christos H. Papadimitriou. 1984. Updates of Relational Views. *J. ACM* 31, 4 (Sept. 1984), 742–760. <https://doi.org/10.1145/1634.1887>
- [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/10.1145/319732.319740>
- [13] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi. 1983. On the Semantics of Updates in Databases. In *PODS*. 352–365. <https://doi.org/10.1145/588058.588100>
- [14] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2015. A Characterization of the Complexity of Resilience and Responsibility for Self-join-free Conjunctive Queries. *PVLDB* 9, 3 (2015), 180–191. <https://doi.org/10.14778/2850583.2850592>
- [15] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2019. New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins. *arXiv:cs.DB/1907.01129*
- [16] Georg Gottlob, Paolo Paolini, and Roberto Zicari. 1988. Properties and Update Semantics of Consistent Views. *ACM Transactions on Database Systems (TODS)* 13, 4 (1988), 486–524.
- [17] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *PODS*. 31–40.
- [18] Joseph Y. Halpern and Judea Pearl. 2005. Causes and Explanations: A structural-model Approach. Part I: Causes. *Brit. J. Phil. Sci.* 56 (2005), 843–887.
- [19] Melanie Herschel and Mauricio A. Hernández. 2010. Explaining Missing Answers to SPJUA Queries. *PVLDB* 3, 1 (2010), 185–196.
- [20] Melanie Herschel, Mauricio A. Hernández, and Wang Chiew Tan. 2009. Artemis: A System for Analyzing Missing Answers. *PVLDB* 2, 2 (2009), 1550–1553.
- [21] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. 2008. On the provenance of non-answers to queries over extracted data. *PVLDB* 1, 1 (2008), 736–747.
- [22] Arthur M. Keller. 1985. Algorithms for Translating View Updates to Database Updates for Views Involving Selections, Projections, and Joins. In *PODS*. 154–163. <https://doi.org/10.1145/325405.325423>
- [23] Benny Kimelfeld. 2012. A Dichotomy in the Complexity of Deletion Propagation with Functional Dependencies. In *PODS*. 191–202. <https://doi.org/10.1145/2213556.2213584>
- [24] Benny Kimelfeld, Jan Vondrák, and Ryan Williams. 2012. Maximizing Conjunctive Views in Deletion Propagation. *ACM TODS* 37, 4, Article 24 (2012), 37 pages. <https://doi.org/10.1145/2389241.2389243>
- [25] Benny Kimelfeld, Jan Vondrák, and David P. Woodruff. 2013. Multi-tuple Deletion Propagation: Approximations and Complexity. *PVLDB* 6, 13 (2013), 1558–1569. <https://doi.org/10.14778/2536258.2536267>
- [26] Phokion G. Kolaitis and Enela Pema. 2012. A Dichotomy in the Complexity of Consistent Query Answering for Queries with Two Atoms. *Inform. Process. Lett.* 112, 3 (Jan. 2012), 77–85. <https://doi.org/10.1016/j.ipl.2011.10.018>
- [27] Paraschos Koutiris and Dan Suciu. 2014. A Dichotomy on the Complexity of Consistent Query Answering for Atoms with Simple Keys. In *17th International Conference on Database Theory (ICDT)*. 165–176. <https://doi.org/10.5441/002/icdt.2014.19>
- [28] Paraschos Koutiris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. 2015. Query-Based Data Pricing. *J. ACM* 62, 5, Article 43 (Nov. 2015), 44 pages. <https://doi.org/10.1145/2770870>

- [29] Paraschos Koutris and Jef Wijsen. 2017. Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. *ACM Trans. Database Syst.* 42, 2, Article 9 (June 2017), 45 pages. <https://doi.org/10.1145/3068334>
- [30] Paraschos Koutris and Jef Wijsen. 2018. Consistent Query Answering for Primary Keys and Conjunctive Queries with Negated Atoms. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (SIGMOD/PODS '18)*. ACM, New York, NY, USA, 209–224. <https://doi.org/10.1145/3196959.3196982>
- [31] Paraschos Koutris and Jef Wijsen. 2018. Consistent Query Answering for Primary Keys in Logspace. *CoRR* abs/1810.03386 (2018). arXiv:1810.03386 <http://arxiv.org/abs/1810.03386>
- [32] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2010. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *PVLDB* 4, 1 (2010), 34–45.
- [33] Alexandra Meliou, Wolfgang Gatterbauer, Suman Nath, and Dan Suciu. 2011. Tracing data errors with view-conditioned causality. In *SIGMOD*. 505–516.
- [34] Sudeepa Roy, Laurel Orr, and Dan Suciu. 2015. Explaining Query Answers with Explanation-ready Databases. *Proc. VLDB Endow.* 9, 4 (Dec. 2015), 348–359. <https://doi.org/10.14778/2856318.2856329>
- [35] Sudeepa Roy and Dan Suciu. 2014. A Formal Approach to Finding Explanations for Database Queries. In *SIGMOD*. 1579–1590.
- [36] Quoc Trung Tran and Chee-Yong Chan. 2010. How to ConQueR why-not questions. In *SIGMOD*. 15–26.
- [37] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB* 6, 8 (2013), 553–564.

## A INDEPENDENT JOIN PATHS: DETAILS

We give more details on the concept of Independent Join Paths. We start with some intuition by providing examples (Appendix A.1), state our conjecture, and finish by pointing out how this concept could possibly allow an automated search for hardness proofs (Appendix A.2), a prospect we are especially excited about.

### A.1 IJP Examples

We give here examples of IJPs for various queries and earlier hardness reductions, and provide the intuition for our 4 conditions.

**Standard paths.** The first example shows that IJPs contain standard paths (Theorem 27) as a special case.

**EXAMPLE 50 ( $q_{vc}$ ).** Consider our simplest example for an SJ-path implying hardness:  $q_{vc}$  from Fig. 2a. The following database of 3 tuples forms an IJP:

$$D = \{R(1), S(1, 2), R(2)\}$$

- (1) We have  $R(1)$  and  $R(2)$  with  $\{1\} \not\subseteq \{2\}$  and  $\{2\} \not\subseteq \{1\}$ .
- (2)  $R(1)$  and  $R(2)$  each participate in only one witness, which in this case is the same one.
- (3)  $R$  being unary, there can't be any other relation with a strict subset of the constants.
- (4) No exogenous relation.
- (5) The resilience  $\rho(q_{vc}, D) = 1$ , but becomes 0 after removing either  $R(1)$  or  $R(2)$  or both.

**Triads.** The second example shows that any query with a triad can form IJPs. We illustrate with our favorite triangle query.

**EXAMPLE 51 ( $q_{\Delta}$ ).** Consider the triangle query as the simplest example of a non-linear SJ-free query containing a triad (see Fig. 1a). The following database of 7 tuples form an IJP:

$$D = \{R(1, 2), R(4, 2), R(4, 5), S(2, 3), S(5, 3), T(3, 1), T(3, 4)\}$$

- (1) We have  $R(1, 2)$  and  $R(4, 5)$  with  $\{1, 2\} \not\subseteq \{4, 5\}$  and  $\{4, 5\} \not\subseteq \{1, 2\}$ .
- (2)  $R(1, 2)$  only participates in witness  $w_1 = (1, 2, 3)$ , and  $R(4, 5)$  only participates in witness  $w_2 = (4, 5, 3)$ .
- (3) No other relation has a strict subsets of the constants from  $R$
- (4) No exogenous relation.
- (5) The resilience  $\rho(q_{\Delta}, D) = 2$ , but becomes 1 after removing either  $R(1, 2)$ , or  $R(4, 5)$ , or both.

Figure 9 illustrates the 3 joins forming the IJP. The connection to our idea from Fig. 8b now becomes clearer. Also notice that this IJP forms the basic element of our prior hardness proof for triads.

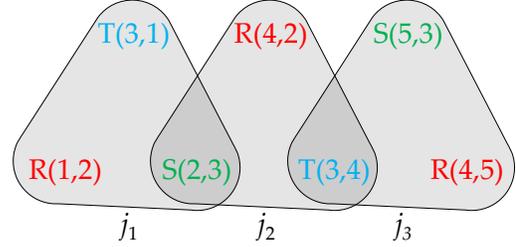


Figure 9: Example 51: IJP for triangle query  $q_{\Delta}$ .

**More complicated IJPs.** The third example uses a more complicated IJP.

**EXAMPLE 52 (MORE COMPLICATED GADGET).** Consider the query

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

Then following database forms an IJP:

$$D = \{A(1), A(4), A(5), A(9), A(13),$$

$$R(1, 2), R(2, 2), R(2, 3), R(3, 3), R(4, 1), R(5, 2),$$

$$R(5, 6), R(6, 7), R(7, 7), R(8, 7), R(9, 8),$$

$$R(1, 10), R(10, 11), R(11, 11), R(12, 11), R(13, 12)\}$$

- (1) We have  $A(9)$  and  $A(13)$ .
- (2)  $A(9)$  only participates in witness  $w_1 = (9, 8, 7)$  and  $A(13)$  only participates in witness  $w_2 = (13, 12, 11)$ .
- (3) No other relation has a strict subset of the constants from  $A$ .
- (4) No exogenous relation.
- (5) The resilience  $\rho(q_{vc}, D) = 4$  with

$$\Gamma = \{R(1, 2), R(2, 2), R(7, 7), R(11, 11)\},$$

but becomes 3 after (i) removing  $A(9)$  with

$$\Gamma = \{A(5), R(1, 2), R(11, 11)\},$$

or (ii) removing  $A(13)$  with

$$\Gamma = \{A(1), R(2, 2), R(7, 7)\},$$

or (iii) removing both with

$$\Gamma = \{A(1), A(5), R(1, 2)\} \text{ or } \Gamma = \{A(1), A(5), R(2, 2)\}.$$

Figure 10 illustrates how these 21 tuples create 8 different joins, representing the IJP. It turns out that this IJP is “hidden” and can be spotted by the careful reader in the crossover part of the variable gadget used in Proposition 39.

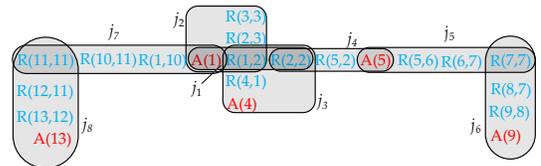


Figure 10: Example 52: IJP for  $z_5$ .

**Condition 4.** We next give one example that illustrates why we need condition 4 of our definition for IJPs. In particular, this query is an example in which two (instead of only one) relation is repeated. We know through a dedicated proof that the complexity of this query is in PTIME. We illustrate a “failed attempt” to create an IJP and point out the problems that would arise if we ignored condition 4.

EXAMPLE 53 (INDEPENDENT PATHS). Consider the following query  $q := A^x(x), R(x), S(x, y), S(z, y), R(z), B^x(z)$  which contains two repeated relations. We investigate the canonical database

$$D = \{R(1), A^x(1), S(1, 2), S(3, 2), R(3), B^x(3)\}$$

and its ability to form an IJP.

- (1) We have  $R(1)$  and  $R(3)$ .
- (2)  $R(1)$  and  $R(3)$  participate in only one witness  $w = (1, 2, 3)$ .
- (3) No other relation has a strict subset of the constants from  $A$ .
- (4) Condition 3 requires that  $B^x(1)$  and  $A^x(3)$  be added to the database, which is currently not the case, and which we ignore for a moment.
- (5) The resilience is 1, and becomes 0 if any tuple is removed.

The crucial condition 4 forces us to add  $B^x(1)$  and  $A^x(3)$  to the database. And then condition 2 and 5 are not true anymore. Addition of these tuples form 2 more joins  $\{R(1), A^x(1), S(1, 2), S(1, 2), R(1), B^x(1)\}$  and  $\{R(3), A^x(3), S(3, 2), S(3, 2), R(3), B^x(3)\}$ , which requires both tuples  $R(1)$  and  $R(3)$  to be removed make the query false.

In other words, the canonical database is not enough to succeed with the reduction from VC (recall Fig. 8b: any two edges incoming and outgoing from vertex  $a$  create addition joins).

## A.2 Toward an automated proof construction

At its core, each IJP can be considered as a set of “canonical databases” or witnesses, which have been appropriately “aligned.” We give the intuition with the triangle query  $q_\Delta$  from Example 51 and Fig. 9.

EXAMPLE 54. Assume we construct three disjoint canonical databases:

$$j_1 : R(1, 2), S(2, 3), T(3, 1)$$

$$j_2 : S(a, b), T(b, 4), R(4, a)$$

$$j_3 : T(c, d), R(d, 5), S(5, d)$$

The total number of constants used is 9, three for each of the three joins.

We can now look at all the possible ways in which these  $n = 9$  constant can be partitioned into nonempty subsets. The answer is given by the Bell number and is 21147 for  $n = 9$ . Exhaustive enumeration over these 21147 cases will also lead to partition

$$\{\{1\}, \{2, a\}, \{3, b, c\}, \{4, d\}, \{5\}\}$$

which is isomorph to the IJP from Fig. 9.

Our Definition 48 now provides a procedure to test that the resulting database indeed forms an IJP.

The more general procedure is now as follows

- (1) for an increasing number of joins  $k = 1, 2, 3, \dots$
- (2) for all possible partitions
- (3) for all pairs of tuples of the same relation that are not dominated
- (4) if an exogenous tuple contains a subset of the constants, then possible add a second tuple
- (5) calculate the minimal VC of the resulting hypergraph under the 4 cases  $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$ , where 0 and 1 mean that a tuple is present or absent, respectively.