Resilience for Regular Path Queries: Towards a Complexity Classification

ANTOINE AMARILLI, Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, France and LTCI, Télécom Paris, Institut polytechnique de Paris, France
WOLFGANG GATTERBAUER, Northeastern University, USA
NEHA MAKHIJA, Northeastern University, USA
MIKAËL MONET, Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

The *resilience problem* for a query and an input set or bag database is to compute the minimum number of facts to remove from the database to make the query false. In this paper, we study how to compute the resilience of Regular Path Queries (RPQs) over graph databases. Our goal is to characterize the regular languages L for which it is tractable to compute the resilience of the existentially-quantified RPQ built from L.

We show that computing the resilience in this sense is tractable (even in combined complexity) for all RPQs defined from so-called *local languages*. By contrast, we show hardness in data complexity for RPQs defined from the following language classes (after reducing the languages to eliminate redundant words): all finite languages featuring a word containing a repeated letter, and all languages featuring a specific kind of counterexample to being local (which we call *four-legged* languages). The latter include in particular all languages that are not *star-free*. Our results also imply hardness for all non-local languages with a so-called *neutral letter*. We last highlight some remaining obstacles towards a full dichotomy.

CCS Concepts: • **Theory of computation → Database theory**; **Formal languages and automata theory**.

Additional Key Words and Phrases: Resilience, Regular Path Queries, Finite Automata

ACM Reference Format:
Antoine Amarilli, Wolfgang Gatterbauer, Neha Makhija, and Mikaël Monet. 2025. Resilience for Regular Path Queries: Towards a Complexity Classification. *Proc. ACM Manag. Data* 3, 2 (PODS), Article 108 (May 2025), 18 pages. https://doi.org/10.1145/3725245

1 Introduction

This paper studies the *resilience problem* which asks, for a Boolean query Q and a given database D (in set or bag semantics) what is a minimum-cardinality set of tuples to remove from D so that Q is no longer satisfied. (In particular: if Q does not hold on D then resilience is zero; if Q holds on D and all subinstances of D then resilience is infinite.) The notion of resilience is motivated by the need to identify how "robust" a query is, especially when the facts of the input database may be incorrect. Specifically, resilience quantifies how many tuples of D must be removed before Q no longer holds, so that higher resilience values mean that Q is less sensitive to tuples of D being removed.

The resilience problem has been introduced for Boolean conjunctive queries and mostly studied for such queries [8, 12, 13, 25]. The key challenge in this line of work is to understand the *data*

Authors' Contact Information: Antoine Amarilli, Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France and LTCI, Télécom Paris, Institut polytechnique de Paris, 91120 Palaiseau, France, antoine.a.amarilli@inria.fr; Wolfgang Gatterbauer Northeastern University, USA, w.gatterbauer@northeastern.edu; Neha Makhija Northeastern University, USA, makhija.n@northeastern.edu; Mikaël Monet, Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France, mikael.monet@inria.fr.

ACM 2836-6573/2025/5-ART108
https://doi.org/10.1145/3725245

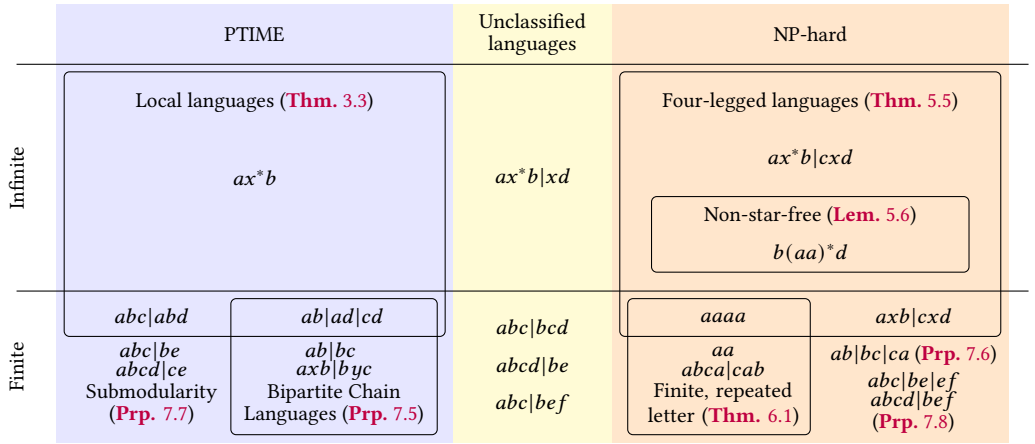


Fig. 1. Overview of our results on the complexity of resilience (all results apply both to bag and set semantics), with some examples of languages. All languages are reduced. For legibility, we omit Proposition 5.7.

complexity of the problem, i.e., its complexity for a fixed query Q, as a function of the input database D. More precisely, the resilience problem can be tractably solved for some queries (e.g., trivial queries), and is known to be NP-hard for some other queries (see, e.g., [25]). Membership in NP always holds whenever query evaluation for Q can be done in polynomial data complexity.

This leads to a natural question: *Can one give a characterization of the queries for which resilience can be tractably computed*? Further, can one show a dichotomy result to establish that resilience is always either in PTIME or NP-complete, depending on the query? This question has remained elusive, already in the case of conjunctive queries (CQs).

In this paper, we study the resilience problem from a different perspective: we focus on graph databases (i.e., databases on an arity-two signature), and on queries which are defined by regular languages, namely, Boolean Regular Path Queries (Boolean RPQs), with no constants and with existentially quantified endpoints. In other words, the query asks whether the database contains a walk whose edge labels forms a word of the regular language, and resilience asks how many database facts must be removed so that no such walk exists. To our knowledge, the only other work which studies this problem is the recent paper of Bodirsky et al. [8], in which the authors use CSP techniques. Their results apply to much more general languages than CQs, and in particular imply a dichotomy on the resilience problem for Boolean RPQs (and also for the broader class of so-called *two-way RPQs* aka 2RPQs). However, their lower bounds do not apply to databases in set semantics. In addition, their dichotomy criterion, while decidable, does not give a intuitive language-theoretic characterization of the tractable cases. We discuss this in more details at the end of Section 2.

In contrast with CQs, RPQs are in certain ways harder to analyze, and in other ways easier. On the one hand, RPQs feature the union operator and the recursion operator; and they inherently correspond to queries with self-joins, because the same relation symbols (corresponding to alphabet letters) may occur multiple times in query matches. On the other hand, RPQs do not allow all features of CQs: they ask for a simple pattern shape (namely, a walk, which importantly is always directed), and they are posed over graph databases (in which all facts have arity two).

Our study of resilience for RPQs is motivated by database applications, but it is also motivated by connections to classical network flow problems. Consider for instance the classical MinCut problem of finding the minimum-cost cut in a flow network. The MinCut problem, allowing multiple sources and sinks without loss of generality, is easily seen to be equivalent to the resilience problem in bag

semantics for the RPQ ax^*b on the alphabet $\{a, b, x\}$, with a-facts of the database corresponding to sources, b-facts to sinks, and x-facts to edges of the network. Hence, the resilience problem for RPQs corresponds to a generalization of MinCut, where the edges of the flow network are labeled and we must cut edges to ensure that no walk in a specific language remains. In the other direction, for many RPQs, we can devise tractable algorithms for resilience by reducing to a MinCut problem on a suitably-constructed flow network. However, resilience is NP-hard for some RPQs, e.g., the finite RPQ aa [13], so these methods do not always apply. Our results aim at determining what are the RPQs for which resilience can be tractably computed (in particular via such methods), and what are the RPQs for which it is intractable.

Contributions and outline. After giving preliminary definitions and stating the problem in Section 2, we start in Section 3 by giving our main tractability result (Theorem 3.3): resilience is tractable for RPQs defined from so-called *local languages* [29, 30]. Those languages are intuitively those that are defined by specifying the possible initial labels of a path, the possible final labels of a path, and the consecutive pairs of labels that are allowed. We show that, for RPQs defined with a local language L, we can compute resilience by reducing to the MinCut problem via a natural product construction. The construction applies to queries both in set and bag semantics, and it is also tractable in combined complexity, running in time $\tilde{O}(|A| \times |\Sigma| \times |D|)$ where A is an automaton representing L, where Σ is the signature, and where D is the input database.

We then turn to intractability results in Sections 4 to 6. In all these results, we assume that the language has been *reduced* to eliminate redundant words, e.g., resilience for $abbc|bb$ and bb is the same because the corresponding (existentially-quantified) Boolean RPQs are equivalent.

We define in Section 4 the notion of a *hardness gadget* for a language L, namely, a specific database to be used as a building block in a reduction. These gadgets are the analogue, in our setting, of the Independent Join Paths (IJPs) used in earlier work [25] to show hardness for CQs. When an RPQ Q has a hardness gadget Γ, we can reduce the minimum vertex cover problem on an undirected graph G to a resilience computation (in set semantics) for Q on a database built using Γ, by replacing the edges of G by a copy of Γ. We illustrate in Section 4 how our gadgets can be used to show hardness for two specific RPQs: $axb|cxd$ (Proposition 4.12), and aa (Proposition 4.1).

We then generalize these two hardness results in turn to apply to a broader class of languages. First, in Section 5, we introduce the notion of a *four-legged language*, as a language featuring a specific kind of counterexample to being local. More precisely, the local languages of Section 3 have the following known equivalent characterization [30]: for any letter x and words $\alpha, \beta, \gamma, \delta$, if a language L contains the words $\alpha x \beta$ and $\gamma x \delta$ but does not contain the "cross-product" word $\alpha x \delta$, then L is not local. A four-legged language is a non-local language that features a counterexample of this form in which all the words $\alpha, \beta, \gamma, \delta$ are required to be non-empty. This covers, for instance, $axb|cxd$, but not $ab|bc$. We show (Theorem 5.5) that resilience is intractable for any four-legged language. This result implies hardness for many languages: it covers all *non-star-free* languages (Lemma 5.6), a well-known class of infinite regular languages; and, along with the tractability of local languages, it implies a dichotomy for languages having a *neutral letter* (Proposition 5.7).

Second, in Section 6, we show that resilience is hard for any finite reduced language containing a word with a repeated letter (Theorem 6.1). This generalizes the case of aa, and shows that self-joins always imply hardness in the following sense: finite RPQs (those equivalent to UCQs), once reduced, have intractable resilience as soon as one of their constituent CQs features a self-join. This contrasts with the setting of CQs, where characterizing the tractable queries with self-joins is still open.

Note that our two hardness results (Theorem 5.5 and Theorem 6.1) are incomparable: languages with a repeated letter are non-local, but they are not all four-legged (e.g., aa); conversely, some four-legged languages have no words with a repeated letter (e.g., $axb|cxd$). Assuming finiteness in

Theorem 6.1 is also crucial, as ax^*b features words with repeated letters and still enjoys tractable resilience. The proof of Theorem 6.1 is our main technical achievement: the proof relies on Theorem 5.5 and uses different gadgets depending on which additional words the language contains.

Section 7 explores how our results do not classify all RPQs. We focus on finite RPQs: the remaining open cases are then the RPQs which have no words with repeated letters, are not local, but are not four-legged. We introduce the class of chain queries (e.g., $axb|byc$) to cover some of these RPQs, and show tractability (and reducibility to MinCut) assuming a bipartiteness condition. We also study the RPQ $abc|be$ and variants thereof, and show that it enjoys tractable resilience via techniques from *submodular optimization* (Proposition 7.7), which seems new in this context. This query (i.e., $R(x,y), S(y,z), T(z,u) \vee S(x,y), U(y,z)$ as a UCQ), and variants thereof, are currently the only known tractable RPQs with no known reduction to MinCut. We conclude by showing that some other unclassified RPQs are hard, and by discussing some remaining open cases.

We finally conclude in Section 8. For lack of space, most proofs are deferred to the appendix of the arXiv version [2].

2 Preliminaries and Problem Statement

Formal languages. Let Σ be an *alphabet*, consisting of *letters*. A *word* over Σ is a finite sequence of elements of Σ. We write ϵ for the empty word. We write Σ^* for the set of words over Σ, and write $\Sigma^+ := \Sigma^* \setminus \{\epsilon\}$. We use lowercase Latin letters such as a, b, x, y to denote letters in Σ, and lowercase Greek letters like $\alpha, \beta, \gamma, \tau$ to denote words in Σ^*. The *concatenation* of two words α, β is written $\alpha\beta$. We write $|\alpha|$ for the length of a word. A word α is an *infix* of another word β if β is of the form $\delta\alpha\gamma$, and it is a *strict infix* if moreover $\delta\gamma \neq \epsilon$. If $\beta = \alpha\gamma$ then α is a *prefix* of β, and a *strict prefix* if $\gamma \neq \epsilon$. Likewise if $\beta = \delta\alpha$ then α is a *suffix* of β, and a *strict suffix* if $\delta \neq \epsilon$.

A *language* L is a (possibly infinite) subset of Σ^*. For L, L' two languages, we write their *concatenation* as $LL' := \{\alpha\beta \mid \alpha \in L, \beta \in L'\}$. We mainly focus in this work on *regular languages*, also known as *rational languages*, which can be represented by *finite state automata*. We use the following automaton formalisms, which recognize precisely the regular languages [30]: *deterministic finite automata* (DFA), *nondeterministic finite automata* (NFA), and *NFAs with ϵ-transitions* (ϵNFA, also called *automata with spontaneous transitions*).

An ϵNFA $A = (S, I, F, \Delta)$ is a finite set S of *states*, a set $I \subseteq S$ of *initial states*, a set $F \subseteq S$ of *final states*, and a *transition relation* $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$. Let $\alpha = a_1 \cdots a_m$ be a word. A *configuration* for A on α is an ordered pair (s, i) for $s \in S$ and $0 \leq i \leq m$. A *run* of A on α is a sequence of configurations $(s_0, i_0), \ldots, (s_\ell, i_\ell)$ with $\ell \geq m$ such that $s_0 \in I$ is initial, $i_0 = 0$, $i_\ell = m$, and for each $0 \leq j < \ell$ we have either $i_{j+1} = i_j$ and $(s_j, \epsilon, s_{j+1}) \in \Delta$ (an ϵ-transition) or $i_{j+1} = i_j + 1$ and $(s_j, a_{i_j+1}, s_{j+1}) \in \Delta$ (a letter transition). The run is *accepting* if $s_\ell \in F$. The *language of* A, denoted $\mathcal{L}(A)$, is the set of words having an accepting run: we say that A *recognizes* $\mathcal{L}(A)$. An example ϵNFA A_3 is depicted in Figure 2c, and an example accepting run of A_3 on the word ad is $(s_1, 0), (s_2, 1), (s_4, 1), (s_5, 2)$. The *size* $|A|$ of the ϵNFA $A = (S, I, F, \Delta)$ is $|S| + |\Delta|$, its total number of states and transitions.

An NFA is an ϵNFA which does not feature ϵ-transitions, and a DFA is an NFA having exactly one initial state and having at most one transition (s, a, s') in Δ for each $s \in S$ and $a \in \Sigma$.

We also use *regular expressions* to denote regular languages: we omit their formal definition, see [30]. We write $|$ for the union operator and $*$ for the Kleene star operator.

Databases, queries, resilience. A *graph database* on Σ, or simply *database*, is a set of Σ-labeled *edges* of the form $D \subseteq V \times \Sigma \times V$, where V is the set of *nodes* of D: we also call V the *domain* and write it as $\mathrm{Adom}(D)$. Notice that a graph database can equivalently be seen as a relational database on an arity-two signature Σ. We also call *facts* the edges (v, a, v') of D, and write them as $v \xrightarrow{a} v'$.

We call v the *tail* of $v \xrightarrow{a} v'$ and v' its *head*. A *Boolean query* is a function Q that takes as input a database D and outputs $Q(D) \in \{0, 1\}$. We write $D \models Q$, and say that Q is *satisfied by* D, when $Q(D) = 1$, and write $D \not\models Q$ otherwise. In this paper we study the *resilience* of queries:

Definition 2.1 (Resilience)**.** *Let D be a database and Q a Boolean query. The* resilience of Q on D, *denoted* $\mathsf{RES}_{\text{set}}(Q, D)$, *is defined*[1] *as* $\mathsf{RES}_{\text{set}}(Q, D) := \min_{\substack{D' \subseteq D \\ \text{s.t. } D \setminus D' \not\models Q}} |D'|$.

In other words, the resilience is the minimum number of facts to remove from D so that Q is not satisfied. A subset of facts $D' \subseteq D$ such that $D \setminus D' \not\models Q$ is called a *cut* (or a *contingency set*) *of D for Q*. Notice that if $D \not\models Q$, then $\emptyset$ is a cut and we have[2] $\mathsf{RES}_{\text{set}}(Q, D) = 0$.

Regular Path Queries. From a language L over Σ we now define a query over graph databases that tests the existence of walks labeled by words of L. Given a database D, a *walk* in D is a (possibly empty) sequence of consecutive edges $v_1 \xrightarrow{a_1} v_2, v_2 \xrightarrow{a_2} v_3, \ldots, v_k \xrightarrow{a_k} v_{k+1}$ in D. Note that the a_i and v_i are not necessarily pairwise distinct. We say that the walk is *labeled* by the word $\alpha = a_1 \cdots a_k$, and call it an *α-walk*. We also say that the walk *contains* a word $\beta \in \Sigma^*$ if β is an infix of α. An *L-walk* in D is then an α-walk for some $\alpha \in L$. We define the Boolean query Q_L as: $Q_L(D) = 1$ iff there is some L-walk in D. In particular, if $\epsilon \in L$ then $Q_L(D) = 1$ on every database D (including $D = \emptyset$). A *regular path query* (RPQ) is then a Boolean query of the form Q_L for L a regular language.

Observe that Q_L does not change if we remove from L all words β such that some strict infix α of β is also in L: indeed if D contains a β-walk then it also contains an α-walk. Formally, the *reduced language of L* is $\mathtt{reduce}(L) := \{\alpha \in L \mid \text{no strict infix of } \alpha \text{ is in } L\}$. We then have that Q_L and $Q_{\mathtt{reduce}(L)}$ are the exact same queries. The reduced language of L is also called the *infix-free sublanguage of L* in [6]: if L is regular then so is $\mathtt{reduce}(L)$. We say that L is *reduced* if $L = \mathtt{reduce}(L)$.

Bag semantics. We also consider the resilience problems on *bag graph databases*, by opposition with *set graph databases* which are the databases defined so far. A *bag (graph) database* consists of a database D together with a *multiplicity function* $\mathtt{mult} : D \to \mathbb{N}_{>0}$; it can equivalently be seen as a multiset of facts. In the input, the multiplicities are expressed as integers written in binary. A Boolean query Q is satisfied by a bag database $(D, \mathtt{mult})$ iff D satisfies Q, i.e., the multiplicities of facts are not seen by queries. A contingency set for Q and D is then a subset $D' \subseteq D$ such that $D \setminus D' \not\models Q$, and the *resilience of Q on D* is: $\mathsf{RES}_{\text{bag}}(Q, D) := \min_{\substack{D' \subseteq D \\ \text{s.t. } D \setminus D' \not\models Q}} \sum_{e \in D'} \mathtt{mult}(e)$, i.e., fact multiplicities represent costs.

Networks and cuts. Most of our resilience algorithms work by reducing to the problem of finding a *minimum cut of a flow network* [10]. Recall that a *flow network* consists of a directed graph $\mathcal{N} = (V, t_{\text{source}}, t_{\text{target}}, E, c)$ with V being the vertices, $t_{\text{source}}, t_{\text{target}} \in V$ being special vertices called the *source* and the *target* respectively, $E \subseteq V \times V$, and $c : E \to \mathbb{R}_+ \cup \{+\infty\}$ giving the *capacity* of each edge. For $E' \subseteq E$, we define the *cost* of E' as $c(E') := \sum_{e \in E'} c(e)$, and denote $\mathcal{N} \setminus E'$ as the network in which all edges from E' are removed. A *cut* is a set of edges $E' \subseteq E$ such that there is no directed path from t_{source} to t_{target} in $\mathcal{N} \setminus E'$. The *min-cut problem*, written MinCut, takes as input a flow network and returns the minimum cost of a cut. From the *max-flow min-cut theorem* [11] and Menger's theorem [28], we know that the min-cut problem can be solved in PTIME. More precisely, practical algorithms can run in $O(|V|^2\sqrt{|E|})$ [1], and recent theoretical algorithms run in $\tilde{O}(|\mathcal{N}|)$ [16], where $|\mathcal{N}| := |V| + |E|$ and where $\tilde{O}$ hides polylogarithmic factors.

[1] By convention, if Q is satisfied by all subsets of D, then the minimum is defined to be $+\infty$.
[2] We deviate from the original definition in [12] which left resilience undefined for the case of $D \not\models Q$.

Problem statement. In this paper, we study the *resilience problem* for queries Q_L defined by languages L; most of our results are for regular languages. The *resilience problem* for L, written $\mathsf{RES}_{\mathrm{set}}(L)$, is the following: given as input a database D and $k \in \mathbb{N}$, decide if $\mathsf{RES}_{\mathrm{set}}(Q_L, D) \leq k$. The problem $\mathsf{RES}_{\mathrm{bag}}(L)$ in bag semantics is defined analogously. Note that $\mathsf{RES}_{\mathrm{set}}(L)$ easily reduces to $\mathsf{RES}_{\mathrm{bag}}(L)$ by giving multiplicity of 1 to each fact; however, in the setting of CQs, some conjunctive queries Q_0 are known for which $\mathsf{RES}_{\mathrm{set}}(Q_0)$ is in PTIME, but $\mathsf{RES}_{\mathrm{bag}}(Q_0)$ is NP-hard [25, Table 1] (even when the multiplicities of the input bag database are written in unary).

Our definitions correspond to the *data complexity* perspective, where we have one computational problem per language L and where the complexity is measured as a function of $|D|$ only. We mostly focus on data complexity, but most of our upper bounds also apply in *combined complexity*, i.e., when both the database D and an automaton A representing L are given as input.

It is easy to see that, for any regular language L, the problem $\mathsf{RES}_{\mathrm{bag}}(L)$, hence $\mathsf{RES}_{\mathrm{set}}(L)$, is in NP (see [2, Appendix A]). However, the NP upper bound is not always tight: there are languages L (e.g., $L = a|b$) such that $\mathsf{RES}_{\mathrm{bag}}(L)$ is in PTIME. The goal of our paper is to characterize the languages L for which $\mathsf{RES}_{\mathrm{set}}(L)$ and $\mathsf{RES}_{\mathrm{bag}}(L)$ are in PTIME, and give efficient algorithms for them.

Connection to VCSPs. From the recent work of Bodirsky et al. [8] using earlier results on Valued Constraint Satisfaction Problems (VCSP), the following dichotomy on $\mathsf{RES}_{\mathrm{bag}}$ is already known:

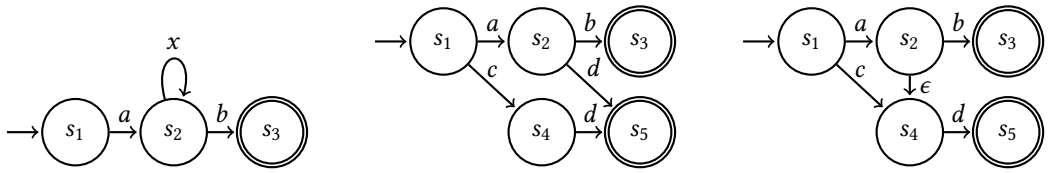
Theorem 2.2 ([8, Remark 8.12])**.** *Let L be a regular language. The problem $\mathsf{RES}_{\mathrm{bag}}(L)$ is either in PTIME or it is NP-complete.*

We point out that this result can be extended to so-called *two-way RPQs*, and to the case where some relations are declared *exogenous* (i.e., they cannot be part of a contingency set). Hence, as discussed in the introduction, Theorem 2.2 gives a complete dichotomy across all regular languages, but it also has some shortcomings. (1.) First, the VCSP approach gives no explicit characterization of the class L of tractable languages, or efficient bounds on the complexity of testing whether a given L is PTIME or not: the criterion in [8], using [22], can be computationally checked but takes time exponential in a representation of the dual of L. (2.) Second, in tractable cases, the VCSP approach does not give good bounds on the degree of the polynomial (unlike, e.g., explicit reductions to MinCut). We remark that difficulties (1.) and (2.) also occur in works studying other problems where VCSP implies a general dichotomy but without explicit characterizations or tight bounds: this is the case for instance in [17]. (3.) Third, the VCSP approach does not shed light on the *combined complexity*, i.e., it does not identify cases that are tractable when both the query and database are given as input. (4.) Fourth, the VCSP approach inherently only applies to bag semantics and not to set semantics, so there is no known analogous dichotomy for $\mathsf{RES}_{\mathrm{set}}$. Our results will improve on some of these shortcomings, though unlike [8] they do not establish a full dichotomy.

3 Tractability for Local Languages

We now start the presentation of our results on the resilience problem by giving our main tractability result in this section. The result applies to *local languages*[3], a well-known subclass of the regular languages [29, 30], whose definition is recalled below. We show that the resilience problem is tractable for local languages, also in combined complexity, using a specific notion of automata and a product construction to reduce resilience to MinCut. We stress that local languages do not cover all cases of regular languages with tractable resilience: we will show other tractable cases in Section 7. However, in Section 5 we will give a partial converse: resilience is hard for the so-called *four-legged* languages, which have a specific form of counterexample to being local.

[3]We note that local languages were also recently observed to correspond to a tractable class of RPQs for which probabilistic query evaluation is equivalent to source-to-target reliability, see [4].



(a) Local DFA A_1 for ax^*b (b) Local DFA A_2 for $ab|ad|cd$ (c) RO-ϵNFA A_3 for $ab|ad|cd$

Fig. 2. Examples of local automata (Example 3.2) and RO-ϵNFAs (see Example 3.2 and Example 3.7). Initial states are indicated by incoming arrows, final states are doubly circled.

Local languages. Intuitively, a regular language L is *local* if membership to L is described by which letter pairs can occur consecutively and which letters can start or can end words. We use an equivalent characterization [29, Chapter III, Section 5.1] via so-called *local DFAs*:

Definition 3.1 (Local DFA)**.** *A DFA is* local *if, for every letter $a \in \Sigma$, all a-transitions in the automaton have the same target state. A language L is* local *if there is a local DFA A that recognizes L.*

Example 3.2. *The languages ax^*b and $ab|ad|cd$ are local, with local DFAs in Figure 2a and Figure 2b.*

The main result of this section is that $\mathsf{RES}_{\mathrm{bag}}(L)$ is PTIME when L is local:

Theorem 3.3 (Resilience for local languages)**.** *If L is local then $\mathsf{RES}_{\mathrm{bag}}(L)$ is in PTIME. Moreover, the problem is also tractable in combined complexity: given an ϵNFA A that recognizes a local language L, and a database D, we can compute $\mathsf{RES}_{\mathrm{bag}}(Q_L, D)$ in $\tilde{O}(|A| \times |D| \times |\Sigma|)$.*

Note that, in the combined complexity upper bound, the input A is not necessarily a local DFA: it suffices to be given an arbitrary ϵNFA A recognizing L, together with the promise that L is local.

Before proving Theorem 3.3, we remark that it implies a PTIME bound on $\mathsf{RES}_{\mathrm{bag}}(L)$ whenever $\mathtt{reduce}(L)$ is local (as $\mathsf{RES}_{\mathrm{bag}}(L)$ is equivalent to $\mathsf{RES}_{\mathrm{bag}}(\mathtt{reduce}(L))$). For instance, $\mathsf{RES}_{\mathrm{bag}}(L_0)$ for $L_0 = \{a, aa\}$ is in PTIME because $\mathtt{reduce}(L_0) = \{a\}$ is local, even though L_0 is not. In fact, to see if Theorem 3.3 applies to a language L, we can always focus on $\mathtt{reduce}(L)$, for the following reason:

Lemma 3.4 (Reduction preserves locality)**.** *If L is local then so is $\mathtt{reduce}(L)$.*

One related question is whether the criterion of Theorem 3.3 can easily be tested: can we easily test whether a given language L is local? If L is given as a DFA, then the task is in PTIME:

Proposition 3.5 (Locality testing)**.** *Given a DFA A, it is in PTIME to determine if $\mathcal{L}(A)$ is local.*

Note that Proposition 3.5 is about testing if the language $L = \mathcal{L}(A)$ recognized by A is local (i.e., some local DFA for L exists); it is not about testing if A is a local DFA (which is easy in linear time).

If the input language L is given as an NFA, we do not know the complexity of testing if L is local: we show in [2, Appendix G] that it is PSPACE-hard to test if L satisfies a property related to being local. Another question that remains open is to test whether the *reduction* of a given language L is local: we can construct an automaton A' representing $\mathtt{reduce}(L)$ from an automaton A representing L and then study A', but the straightforward construction for A' entails an exponential blowup ([6, Proposition 6]).

Read-Once automata. The proof of Theorem 3.3 is not difficult and is a simple reduction to the MinCut problem. To present it, we need a variant of local automata, which we name *Read-Once (RO) automata*. Intuitively, RO automata strengthen the requirement of local automata by requiring that for each letter $a \in \Sigma$ there is *at most one transition* labeled with a; but in exchange RO automata are not required to be deterministic and allow for (arbitrarily many) ϵ-transitions. Formally:

Definition 3.6 (RO-ϵNFA)**.** *A* read-once ϵNFA *(RO-ϵNFA for short) is an ϵNFA $A = (S, I, F, \Delta)$ such that for every letter $a \in \Sigma$ there is at most one transition of the form (s, a, s') in Δ.*

Example 3.7 (Example 3.2 continued)**.** *The local DFA A_1 in Figure 2a for ax^*b is an RO-ϵNFA, but the local DFA A_2 in Figure 2b for $ab|ad|cd$ is not. An RO-ϵNFA for this language is A_3 (Figure 2c).*

We show that RO-ϵNFAs recognize precisely the local languages (and the presence of ϵ-transitions is essential for this, see [2, Appendix H]):

Lemma 3.8 (RO-ϵNFA = local languages)**.** *A language L is local iff it is recognized by an RO-ϵNFA. Moreover, given an ϵNFA A recognizing a local language $L = \mathcal{L}(A)$, we can build in $O(|A| \times |\Sigma|)$ an RO-ϵNFA recognizing L.*

The point of RO-ϵNFAs is that we can use them to solve resilience with a product construction:

Proof of Theorem 3.3. Given an ϵ-NFA A' for the local language L, we compute an RO-ϵNFA $A = (S, I, F, \Delta)$ for L in $O(|A'| \times |\Sigma|)$ using Lemma 3.8. Let D be the input bag database with fact multiplicities given by `mult`, and let V be the domain of D. Define a network $\mathcal{N}_{D,A}$ as follows:

- The set of vertices is $(V \times S) \cup \{t_{\text{source}}, t_{\text{target}}\}$, with the fresh vertices t_{source} and t_{target} being respectively the source and target of the network.
- There is an edge with capacity $\texttt{mult}(v \xrightarrow{a} v')$ from (v, s) to (v', s') for all $v, v' \in V$ and $s, s' \in S$ and $a \in \Sigma$ such that $v \xrightarrow{a} v' \in D$ and $(s, a, s') \in \Delta$
- There is an edge with capacity $+\infty$ from (v, s) to (v, s') for all $v \in V$ and $s, s' \in S$ such that $(s, \epsilon, s') \in \Delta$.
- There is an edge with capacity $+\infty$ from t_{source} to every vertex of the form (v, s) with $s \in I$, and from every vertex of the form (v, s) with $s \in F$ to t_{target}.

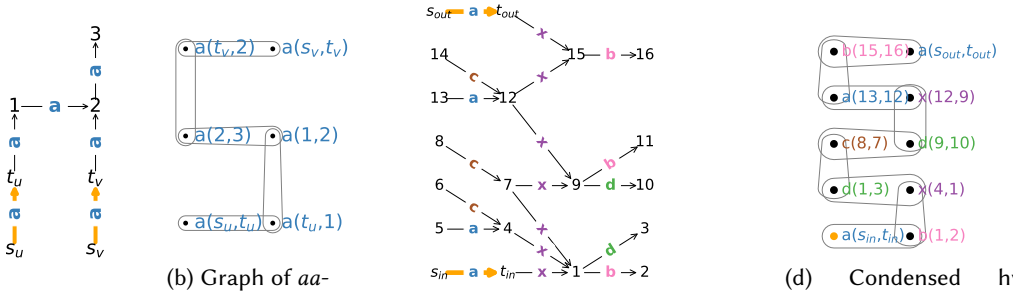
Clearly $\mathcal{N}_{D,A}$ is built in $O(|A| \times |D|)$ from A and D. Moreover, there is a one-to-one correspondence between the contingency sets of Q_L for D and the cuts of $\mathcal{N}_{D,A}$ that have finite cost: intuitively, selecting facts of D in the contingency set corresponds to cutting edges in $\mathcal{N}_{D,A}$, and ensuring that the remaining facts do not satisfy Q_L corresponds to ensuring that no source to target paths remain. Crucially, as A is read-once, the correspondence between facts of D and edges of $\mathcal{N}_{D,A}$ with finite capacity is one-to-one. Further, the sum of multiplicities of the facts of a contingency set is equal to the cost of the corresponding cut, so minimum cuts correspond to minimum contingency sets. We can thus solve MinCut on $\mathcal{N}_{D,A}$ in $\tilde{O}(|\mathcal{N}_{D,A}|)$ [16] to compute the answer to $\texttt{RES}_{\text{bag}}(L)$ in D. □

4 Hardness Techniques and First Results

In this section, we move on to cases where computing resilience is intractable. From now on we only study languages that are reduced, because showing hardness for a reduced language L is sufficient to imply the same for any language L' such that $\texttt{reduce}(L') = L$.

We start the section by showing the NP-hardness of $\texttt{RES}_{\text{set}}(aa)$, by an explicit reduction from the minimum vertex cover problem. We then generalize this reduction and define the notion of *(hardness) gadgets* as databases that can be used to encode graphs and reduce from minimum vertex cover. We show that whenever a language L admits a gadget then $\texttt{RES}_{\text{set}}(L)$ is NP-hard. We finally illustrate these techniques to show the hardness of resilience for the specific language $axb|cxd$.

The notion of gadgets introduced in this section will be used in the next two sections to show the hardness of entire language families. Specifically, we will generalize in Section 5 the hardness of $\texttt{RES}_{\text{set}}(axb|cxd)$ to that of the so-called *four-legged languages*; and generalize in Section 6 the hardness of $\texttt{RES}_{\text{set}}(aa)$ to that of all finite languages featuring a word with a repeated letter.



(a) Completed gadget for *aa*

(b) Graph of *aa*-matches of Figure 3a

(c) Completed gadget for $axb|cxd$

(d) Condensed hypergraph of $axb|cxd$-matches of Figure 3c

Fig. 3. Illustration of hardness gadgets and corresponding condensed hypergraphs of matches. Figures 3a and 3b illustrates Proposition 4.1, while Figures 3c and 3d illustrates Proposition 4.12.

4.1 Hardness of *aa*

Let us first show that resilience is hard for the language *aa*. The result is known from the setting of CQs [13, Prop. 10] by a direct reduction from 3-SAT. We give a self-contained proof where we reduce instead from minimum vertex cover: this proof serves as an intuitive presentation of the more general constructions that come next.

Proposition 4.1. $\mathsf{RES}_{\mathsf{set}}(aa)$ *is NP-hard.*

In other words, given a directed graph G and integer $k \in \mathbb{N}$, it is NP-hard to decide if we can remove k edges of G such that G no longer contains any path of length 2. We show Proposition 4.1 by reducing from the *minimum vertex cover problem*. Recall that a *vertex cover* of an undirected graph $G = (V, E)$ is a subset of vertices $V' \subseteq V$ such that every edge of E is incident to a vertex of V', i.e., for each $e \in E$, we have $e \cap V' \neq \emptyset$. The *vertex cover number* of G is the minimum cardinality of a vertex cover of G. Given an undirected graph G and an integer $k \in \mathbb{N}$, the *minimum vertex cover problem* asks if the vertex cover number of G is at most k. It is an NP-complete problem [20].

Before showing the reduction, we show that the vertex cover number of an undirected graph G can be computed from the vertex cover number of any odd-length subdivision of G. Formally, for $\ell \in \mathbb{N}$, $\ell \geq 1$, an *ℓ-subdivision* of G is an undirected graph G' obtained by replacing each edge of G by a path of length ℓ (i.e., with $\ell - 1$ fresh internal vertices, that are disjoint across edges). We claim:

Proposition 4.2 (Hardness of vertex cover on subdivisions)**.** *Let $\ell \in \mathbb{N}$ be an odd integer. Then for any undirected graph $G = (V, E)$, letting G' be an ℓ-subdivision of G, then the vertex cover number of G' is $k + m(\ell - 1)/2$, where k is the vertex cover number of G and m is the number of edges of G.*

Let us now show the hardness of resilience for the language *aa*:

Proof of Proposition 4.1. We reduce from the vertex cover problem. Let G be an undirected graph and $k \in \mathbb{N}$ be an integer. We start by picking an arbitrary orientation for each edge of G, giving a directed graph $G' = (V, E)$. We then construct a database Ξ in the following way (illustrated in [2, Appendix C.2]). First, for each vertex $u \in V$ we create a fact $s_u \xrightarrow{a} t_u$. Second, for each edge $e = (u, v)$ of E we create the following set D_e of facts: $t_u \xrightarrow{a} t_{e,1}$, $t_{e,1} \xrightarrow{a} t_{e,2}$, $t_{e,2} \xrightarrow{a} t_{e,3}$, $t_v \xrightarrow{a} t_{e,2}$. Together with $s_u \xrightarrow{a} t_u$ and $s_v \xrightarrow{a} t_v$, the facts of D_e form an instance that we call the *completed gadget* D'_e for e: we illustrate D'_e in Figure 3a, renaming $t_{e,i}$ simply to i for brevity. We introduce the notations $F_{e,\text{in}} = s_u \xrightarrow{a} t_u$ and $F_{e,\text{out}} = s_v \xrightarrow{a} t_v$ to refer to the two facts of $D'_e \setminus D_e$, called the

endpoint facts: they are shown in bold orange in Figure 3a. Building Ξ from G is in $O(|G|)$ (as D_e has constant size).

The resilience of aa on Ξ is then equal to the vertex cover number of a 5-subdivision[4] of G. Indeed, define the *graph of aa-matches* on a database D as the undirected graph having one vertex for each fact of D and one edge for each pair of facts of D that forms an aa-path. It is then clear that $\mathsf{RES}_{\mathsf{set}}(aa, \Xi)$ is equal to the vertex cover number of the graph of aa-matches of Ξ. Moreover, the graph of aa-matches of each D'_e is a path of length 5 from vertex $F_{e,\mathrm{in}}$ to vertex $F_{e,\mathrm{out}}$ (see [2, Appendix C.2, Figure 5]). Thus, the graph of aa-matches of Ξ is (isomorphic to) a 5-subdivision of G, and we conclude by Proposition 4.2. □

4.2 Hardness Gadgets

To show the hardness of resilience for other languages than aa, we will build *hardness gadgets*, or *gadgets* for brevity. Gadgets are used as building blocks to reduce from the vertex cover problem, generalizing the idea of Figure 3a from Section 4.1. We first define *pre-gadgets* as databases with two domain elements to which endpoint facts will be attached. We then define the *encoding* Ξ of a directed graph G by gluing together copies of the pre-gadgets following G, as in the proof of Proposition 4.1. We last define *gadgets* as pre-gadgets whose hypergraph of matches of the query Q forms an odd chain. The vertex cover problem on G then reduces to the resilience of Q on Ξ.

Definition 4.3 (Pre-gadget)**.** *A* pre-gadget *is a tuple $\Gamma = (D, t_{\mathrm{in}}, t_{\mathrm{out}}, a)$ consisting of a database D, two distinguished domain elements $t_{\mathrm{in}} \neq t_{\mathrm{out}}$ of D respectively called the* in-element *and* out-element*, and a label $a \in \Sigma$. We require that t_{in} and t_{out} never occur as heads of facts of D. The* completion *of Γ is the database D' obtained from D by adding two fresh distinct domain elements s_{in} and s_{out} and the two facts $s_{\mathrm{in}} \xrightarrow{a} t_{\mathrm{in}}$ and $s_{\mathrm{out}} \xrightarrow{a} t_{\mathrm{out}}$ using letter a.*

Example 4.4. *Figure 3a gives an example completion of a pre-gadget. The in-element and out-element of the pre-gadget are respectively t_u and t_v, and the facts $s_u \xrightarrow{a} t_u$ and $s_v \xrightarrow{a} t_v$ are the facts added in the completion (so the pre-gadget has 4 edges and the completion has 6).*

We next explain how we use gadgets to encode directed graphs (see [2, Figure 5]):

Definition 4.5 (Encoding a graph with a pre-gadget)**.** *Given a directed graph G and a pre-gadget $\Gamma = (D, t_{\mathrm{in}}, t_{\mathrm{out}}, a)$, the* encoding *of G with Γ is the database Ξ obtained from G as follows:*

1. *For each node u in G, create a fact $s_u \xrightarrow{a} t_u$ in Ξ, where s_u and t_u are fresh domain elements.*
2. *For each edge $e = (u, v)$ in G, add to Ξ a copy D_e of D defined by the following isomorphism:*
   - *domain element t_{in} is renamed to t_u and domain element t_{out} is renamed to t_v*
   - *all other elements are renamed to fresh elements called the* internal elements *of D_e.*

Let us observe the following property about the encoding, which is the reason why we require that in-elements and out-elements never occur as the head of facts in pre-gadgets (before completion).

Claim 4.6 (Paths in encodings)**.** *Let G be a directed graph, let $\Gamma = (D, t_{\mathrm{in}}, t_{\mathrm{out}}, a)$ be a pre-gadget, and let Ξ be the encoding of G with Γ. Then no walk of Ξ passes through two internal elements belonging to different copies of D.*

For now, our definition of pre-gadgets is independent of the query. Let us now formalize what we require from query matches in gadgets, in order to reduce from the vertex cover problem. We do this by introducing the *hypergraph of matches*, generalizing the graph of matches from the proof of Proposition 4.1, along with a *condensation* operation to simplify it.

[4] A simpler construction can achieve a 3-subdivision, but our construction will generalize better for subsequent definitions.

4.3 Condensed Hypergraph of Matches

Let L be a language and D a database. A *match* (or witness) of L on D is a set of edges corresponding to an L-walk (as defined in Section 2): formally, each L-walk $e_1, \ldots, e_m$ defines a match which is the set $\{e_1, \ldots, e_m\}$, noting that several L-walks may define the same match. Then:

Definition 4.7 (Hypergraph of matches)**.** *The* hypergraph of matches *of D and L, denoted $\mathcal{H}_{L,D}$, is the hypergraph whose nodes are the facts of D and whose hyperedges are the matches of L on D.*

Recall that a *hitting set* (also called *hypergraph transversal* [15]) of a hypergraph $\mathcal{H} = (V, E)$ is a subset $V' \subseteq V$ such that $V' \cap e \neq \emptyset$ for every $e \in E$. Thus, hitting sets are a generalization of vertex covers to hypergraphs. It is clear from the definition of resilience that $\mathsf{RES}_{\text{set}}(Q_L, D)$ is equal to the minimum size of a hitting set of the hypergraph of matches $\mathcal{H}_{L,D}$. We now introduce two rewriting rules, called the *condensation rules*, that can be applied to any hypergraph $\mathcal{H} = (V, E)$ to simplify it without affecting the minimum size of hitting sets. For $v \in V$, write $E(v) = \{e \in E \mid v \in e\}$. The condensation rules are:

Edge-domination. If there are $e, e' \in E$ with $e \neq e'$ and $e \subseteq e'$, we can remove e' to obtain $\mathcal{H}' = (V, E \setminus \{e'\})$. Intuitively, hitting sets must intersect e so they automatically intersect e'.

Node-domination. If there are $v, v' \in V$ such that $E(v) \subseteq E(v')$, we can remove v to obtain $\mathcal{H}' = (V \setminus \{v\}, \{e \setminus \{v\} \mid e \in E\})$. Intuitively, we can always replace v by v' in hitting sets.

We say that $\mathcal{H}'$ is a *condensed hypergraph* of $\mathcal{H}$ if $\mathcal{H}'$ can be obtained from $\mathcal{H}$ by applying some sequence of condensation rules (note that we do *not* require that $\mathcal{H}'$ cannot be further simplified using the rules). One can easily see that the rules preserve the minimum size of a hitting set:

Claim 4.8 (Condensation preserves minimum size of hitting sets)**.** *Let $\mathcal{H}'$ be a condensed hypergraph of $\mathcal{H}$. Then the minimum size of a hitting set of $\mathcal{H}$ and $\mathcal{H}'$ are the same.*

By Claim 4.8, we know that $\mathsf{RES}_{\text{set}}(Q_L, D)$ is equal to the minimum size of a hitting set of any condensed hypergraph of $\mathcal{H}_{L,D}$, so we can always apply condensation rules in hardness proofs.

We can now define the requirement for a pre-gadget $\Gamma = (D, t_{\text{in}}, t_{\text{out}}, a)$ to be a gadget[5]: we require that the hypergraph of matches $\mathcal{H}_{L,D'}$ of L on the completion D' of Γ can be condensed into an *odd path*. Formally, let $\mathcal{H} = (V, E)$ be a hypergraph and $u, v \in V$. We call $\mathcal{H}$ an *odd path from u to v* if all hyperedges of $\mathcal{H}$ have size 2 and the corresponding graph is an odd path from u to v; i.e., there is a sequence $u = w_1, \ldots, w_{2k} = v$ such that $E = \{\{w_i, w_{i+1}\} \mid 1 \leq i < 2k\}$.

Definition 4.9 (Gadget)**.** *A* gadget *for language L is a pre-gadget $\Gamma = (D, t_{\text{in}}, t_{\text{out}}, a)$ that satisfies the following condition: letting D' be the completion of Γ (see Definition 4.3) where we add two a-facts F_{in} and F_{out} from fresh elements to t_{in} and t_{out} respectively, there exists a condensation of the hypergraph of matches $\mathcal{H}_{L,D'}$ of L on D' which is an odd path from F_{in} to F_{out}.*

Example 4.10. *The hardness pre-gadget whose completion is pictured in Figure 3a is a gadget for aa, as can be seen on the hypergraph of matches of Figure 3b (with no condensation rules applied).*

We claim that the existence of a gadget implies that the resilience problem is hard, namely:

Proposition 4.11 (Gadgets imply hardness)**.** *Let L be a reduced language. If there is a gadget Γ for L, then $\mathsf{RES}_{\text{set}}(L)$ is NP-hard.*

Proof sketch. The proof follows the same idea as that of Proposition 4.1. We reduce from the vertex cover problem: let G' be an undirected graph and G be a directed graph obtained by choosing an arbitrary orientation for the edges of G'. We build in linear time the encoding Ξ of G by Γ. We

[5]Note that a notion generalizing gadgets has been introduced under the name of *Independent Join Paths* in [25], where it is used to show the hardness of resilience of conjunctive queries.

use the oracle to $\mathsf{RES}_{\mathrm{set}}(L)$ to obtain $r := \mathsf{RES}_{\mathrm{set}}(Q_L, \Xi)$, which is equal to the minimum size of a hitting set of the hypergraph $\mathcal{H}_{L,\Xi}$ of matches of L on Ξ. Using Claim 4.6 and the fact that Γ is a gadget, we show that there exists a condensation $\mathcal{H}'$ of $\mathcal{H}_{L,\Xi}$ that has only hyperedges of size 2 and which (when seen as an undirected graph) is an odd-length subdivision G'' of G'. By Claim 4.8, $r = \mathsf{RES}_{\mathrm{set}}(Q_L, \Xi)$ is equal to the minimum size of a hitting set of $\mathcal{H}'$, i.e., r is exactly the vertex cover number of G'', from which we recover the vertex cover number of G' via Proposition 4.2. □

Hence, from now on, all hardness results will be shown by simply exhibiting hardness gadgets for the languages in question. We also wrote an implementation [3] to automatically check some correctness properties on our gadgets.

4.4 Hardness for $axb|cxd$

We illustrate the use of gadgets by building one for the language $axb|cxd$:

Proposition 4.12. $\mathsf{RES}_{\mathrm{set}}(axb|cxd)$ *is NP-hard.*

Proof. We use the gadget Γ for the language $L = axb|cxd$ whose completion D' is depicted in Figure 3c. Indeed, Γ is clearly a pre-gadget. Further, we explain in [2, Appendix C.6] why the hypergraph of matches $\mathcal{H}_{L,D'}$ of L on D' (shown in in [2, Figure 6]) can be condensed to the hypergraph of Figure 3d, which is an odd path. Thus, Γ is a gadget so Proposition 4.11 concludes. □

5 Hardness of Four-Legged Languages

In the previous section, we have presented hardness gadgets, and shown the hardness of resilience for the two specific languages aa and $axb|cxd$. In this section we generalize the latter result to a subclass of the reduced non-local languages, called the *four-legged languages*: these languages intuitively feature a specific kind of counterexample to locality. We show that every four-legged language admits a gadget, so that resilience for such languages is always NP-hard by Proposition 4.11. We give two consequences of this result: it implies that resilience is hard for all *non-star-free* regular languages; and it also implies a dichotomy on resilience for languages featuring a *neutral letter*.

5.1 Four-Legged Languages

Recall the notion of local languages (Definition 3.1) from Section 3, which hinges on recognizability by certain automata (namely, local DFAs). We now give a known equivalent language-theoretic characterization of local languages, that we call being *letter-Cartesian*:

Definition 5.1 (Letter-Cartesian [30])**.** *A language L is* letter-Cartesian *if the following implication holds: for every letter $x \in \Sigma$ and words $\alpha, \beta, \gamma, \delta \in \Sigma^*$, if $\alpha x \beta \in L$ and $\gamma x \delta \in L$, then $\alpha x \delta \in L$.*

Note that, by exchanging the roles of α, β and γ, δ, if $\alpha x \beta \in L$ and $\gamma x \delta \in L$ then also $\gamma x \beta \in L$. Hence the name *letter-Cartesian*: for each letter $x \in \Sigma$, the words of L containing an x can be expressed as a Cartesian product between what can appear left of an x and what can appear right of an x. Formally, we have $L = \bigcup_{x \in \Sigma} L_x x R_x$ for some $L_x, R_x \subseteq \Sigma^*$. It is known that letter-Cartesian languages are precisely the local languages: see [30, Chp. II, Ex. 1.8, (d)] for a statement without proof, or [2, Proposition B.7] for a self-contained proof.

Example 5.2. *The local languages of Example 3.7 are letter-Cartesian. Take now $L = aa$. With $x := a$, $\alpha := a$, $\beta := \epsilon$, $\gamma := \epsilon$, $\delta := a$, we see that L is not letter-Cartesian (as $aaa \notin L$), hence it is not local.*

If a language L is not letter-Cartesian, by definition there are $x \in \Sigma$ and $\alpha, \beta, \gamma, \delta \in \Sigma^*$ such that $\alpha x \beta \in L$ and $\gamma x \delta \in L$ but $\alpha x \delta \notin L$. We then define the *four-legged languages* as those reduced languages that feature such a violation in which none of the words α, β, γ, and δ is empty. Formally:

Definition 5.3 (Four-legged)**.** *A language L is* four-legged *if it is reduced and has a letter $x \in \Sigma$ (the* body*) and four non-empty words $\alpha, \beta, \gamma, \delta \in \Sigma^+$ (the* legs*) with $\alpha x \beta \in L$ and $\gamma x \delta \in L$ but $\alpha x \delta \notin L$.*

Example 5.4. *The languages $axb|cxd$ (from Proposition 4.12) and $axb|cxd|cxb$ are non-local and four-legged. The languages aa (from Proposition 4.1) and $ab|bc$ are non-local but not four-legged.*

Note that four-legged languages are never local, because they are not letter-Cartesian. The main result of this section is that $\mathsf{RES}_{\mathrm{set}}(L)$ is always NP-hard when L is four-legged; formally:

Theorem 5.5 (Hardness of four-legged languages)**.** *If L is four-legged then $\mathsf{RES}_{\mathrm{set}}(L)$ is NP-hard.*

Proof sketch. Let L be a four-legged language, with body $x \in \Sigma$ and legs $\alpha, \beta, \gamma, \delta \in \Sigma^+$ so that $\alpha x \beta \in L$ and $\gamma x \delta \in L$ but $\alpha x \delta \notin L$. We first show that, up to changing the legs, we can ensure that no infix of $\alpha x \delta$ is in L. Then we distinguish two cases, depending on whether some infixes of $\gamma x \beta$ are in L or not, and exhibit a gadget for each case. See [2, Appendix D.1] for details. □

Note that Theorem 5.5 is not restricted to regular languages and applies to arbitrary languages (even when resilience is not in NP). Next, we spell out two consequences of the result.

5.2 Implications of Hardness for Four-Legged Languages

Hardness for non-star-free languages. We say that a regular language L is *star-free* if there exists a $k > 0$ such that, for every $\rho, \sigma, \tau \in \Sigma^*$, for every $m \geq k$, we have $\rho \sigma^k \tau \in L$ iff $\rho \sigma^m \tau \in L$. Star-free languages have many equivalent characterizations (see, e.g., [29]), and if a language L is star-free then so is $\mathtt{reduce}(L)$ (see [2, Appendix D.2]). We can show that being star-free is a necessary condition to enjoy tractable resilience, because non-star-free reduced regular languages are four-legged and hence resilience for them is NP-hard by Theorem 5.5:

Lemma 5.6 (Hardness of non-star-free languages)**.** *Let L be a regular language which is reduced and non-star-free. Then L is four-legged.*

Dichotomy for languages with a neutral letter. A second consequence of Theorem 5.5 concerns languages L with a *neutral* letter, i.e., a letter $e \in \Sigma$ such that inserting or deleting e anywhere does not change the membership of words to L. Formally, e is neutral for L if for every $\alpha, \beta \in \Sigma^*$ we have $\alpha\beta \in L$ iff $\alpha e \beta \in L$. In formal language theory, one often assumes the existence of a neutral letter as a technical assumption on languages [7, 23]. Under this assumption, we can show:

Proposition 5.7 (Dichotomy for languages with neutral letters)**.** *Let L be a language with a neutral letter. If $\mathtt{reduce}(L)$ is local then $\mathsf{RES}_{\mathrm{bag}}(L)$ is PTIME, otherwise $\mathsf{RES}_{\mathrm{set}}(L)$ is NP-hard.*

Proof sketch. If $\mathtt{reduce}(L)$ is local then we have tractability thanks to Theorem 3.3. Otherwise, we can show that $\mathtt{reduce}(L)$ is either four-legged or contains a word of the form aa (or both), so that at least one of Proposition 4.1 and Theorem 5.5 applies. □

The dichotomy above relies on the presence of a neutral letter, so it does not apply to all languages. In particular, it does not apply at all to finite languages, so we focus on them from now on.

6 Hardness of Finite Languages Having Repeated Letters

We now conclude the presentation of our main hardness results by showing that resilience is hard for a class of languages that generalizes aa (Proposition 4.1). We say that a word α has a *repeated letter* if we have $\alpha = \beta a \gamma a \delta$ for some $a \in \Sigma$ and $\beta, \gamma, \delta \in \Sigma^*$. We show that, for reduced and finite languages, resilience is NP-hard whenever the language contains a word with a repeated letter:

Theorem 6.1 (Hardness with repeated letters)**.** *Let L be a reduced finite language that contains a word with a repeated letter. Then $\mathsf{RES}_{\mathrm{set}}(L)$ is NP-complete.*

Proof sketch. The proof is rather technical. We use the finiteness of L to select a word with a repeated letter of the form $\beta a \gamma a \delta$ where the gap between repeated letters is maximal. A long case distinction then allows us to deduce that either L is four-legged (in which case the result follows from Theorem 5.5), or we can show that one of several gadgets applies. □

Connections to UCQs and self-joins. Observe that finite languages L correspond to the case of RPQs that are in fact unions of conjunctive queries (UCQs). The UCQs obtained from RPQs have a specific form: they are posed on an arity-two signature corresponding to the alphabet Σ, and their constituent conjunctive queries (CQs) are directed paths corresponding to words of the language L. In this sense, repeated letters in words correspond to self-joins in CQs. Thus, Theorem 6.1 implies that, for those UCQs that correspond to RPQs, the resilience problem is NP-hard unless every CQ is self-join-free. This contrasts with the setting of general UCQs, where resilience is sometimes tractable even in the presence of self-joins [13]. Specifically, in that setting, the following Boolean CQ on an arity-two signature is known to be in PTIME [13] despite having two self-joins: $Q :- A(x), B(x), R(x, y), R(z, y), A(z), C(z)$.

Connections to other language classes. We now explain how Theorem 6.1 relates to the languages discussed so far. We first perform a sanity check: such languages are never local.

Lemma 6.2. *Let L be a finite language containing a word with a repeated letter. Then L is not local.*

Note that the finiteness hypothesis is essential, e.g., $L = ax^*b$ contains words with repeated letters but it is local (see Example 3.2). Of course it is also essential that L is reduced, e.g., $L = a|aa$ contains a word with a repeated letter but its reduction $\texttt{reduce}(L) = a$ is local.

Further notice that there is no converse to Lemma 6.2: there are finite reduced non-local languages that feature no words with repeated letters and for which resilience is hard. This is for instance the case of the four-legged language $axb|cxd$ of Proposition 4.12. Thus, the languages covered by Theorem 6.1 and Theorem 5.5 are incomparable: they each generalize Proposition 4.1 and Proposition 4.12, respectively, without implying the other result. (However, the proof of Theorem 6.1 relies on Theorem 5.5.) These two results also do not cover all intractable cases, as we will now see.

7 Additional Complexity Results

We have shown the tractability of resilience for local languages (Theorem 3.3), and shown hardness for four-legged languages (Theorem 5.5) and for finite reduced languages featuring a word with a repeated letter (Theorem 6.1). Our results imply a dichotomy over languages with a neutral letter (Proposition 5.7), but not in general. In this section, we classify some languages not covered by these results, and highlight remaining open cases. We focus on languages that are finite and reduced.

We first illustrate two cases of finite languages that are tractable but not local. One are those we call *bipartite chain languages*, such as $ab|bc$. For them, we can compute resilience via a variant of the product construction of Theorem 3.3, simply by "reversing" some matches. Another is a class of languages (including for instance $abc|be$) which we show tractable via *submodular minimization*. These are the only tractable case where we do not know how to reduce to a MinCut problem. We conclude by showing examples of queries not covered by our previous hardness results but for which resilience can be shown intractable, and finally exhibit queries that remain unclassified.

Tractability of Bipartite Chain Languages. Let us define the *chain languages*:

Definition 7.1 (Chain languages)**.** *A language L is a* chain language *if:*

(1) *No word in L contains a repeated letter.*
(2) *Each word of L of length at least 2 may only share its first or last letter with other words in L: formally, for each word $\alpha = a\gamma b$ in L, no letter of γ occurs in another word of L.*

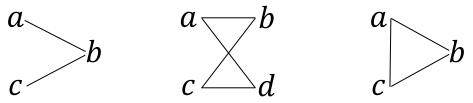


Fig. 4. Illustrations of the endpoint graphs for the three languages mentioned in Example 7.3.

Note that chain languages are always finite, because they contain at most $|\Sigma|$ words of length > 2: indeed, each word of length 3 or more contains intermediate letters which cannot occur in another word. We will focus on chain languages satisfying an extra condition, called being *bipartite*:

Definition 7.2 (Endpoint graph, BCL)**.** *The* endpoint graph *of a language L is the undirected graph (Σ, E) having as vertices the letters of Σ, and having an edge $\{a, b\}$ for $a \neq b$ in Σ iff a and b are the endpoints of some word of L of length at least 2 (i.e., L contains a word of the form $a\alpha b$ or $b\alpha a$).*

A chain language is a bipartite chain language *(BCL) if its endpoint graph is bipartite.*

Example 7.3 (Figure 4)**.** *The languages $ab|bc$ (from Example 5.4) and $axyb|bztc|cd|dea$ are BCLs; the language $ab|bc|ca$ is a chain language but not a BCL. None of these languages are local.*

Notice that the definition of BCLs is *symmetric*: any word α of L can be replaced by the mirror word of α and the resulting language is still a BCL. Also notice that the class of BCLs is *incomparable* to local languages (even finite local languages): indeed, ax^*b and $axb|axc$ are local languages but not BCLs, and Example 7.3 shows BCLs that are not local. Further note that BCLs containing no word of length less than 2 are always reduced. Further, we have the immediate analogue of Lemma 3.4: the reduced language of a BCL is always a BCL. More generally:

Lemma 7.4. *For any BCL L, any $L' \subseteq L$ is also a BCL.*

We can show that resilience is tractable for all BCLs, even in combined complexity:

Proposition 7.5 (Bipartite chain languages are tractable)**.** *If L is a BCL then $\mathsf{RES}_{\mathrm{bag}}(L)$ is in PTIME. Moreover, the problem is also tractable in combined complexity: given an ϵNFA A that recognizes a BCL language L, and a database D, we can compute $\mathsf{RES}_{\mathrm{bag}}(Q_L, D)$ in $\tilde{O}(|A| \times |D|^2 \times |\Sigma|^2)$.*

Proof sketch. We partition the letters of Σ based on the bipartition of the endpoint graph. We then build a flow network like for Theorem 3.3, but we reverse some words depending on the sides of the partition in which their endpoints fall. □

As for chain languages L that are not bipartite, we conjecture that $\mathsf{RES}_{\mathrm{set}}(L)$ is always NP-hard. We leave this open, but show hardness for the specific non-bipartite chain language of Example 7.3:

Proposition 7.6. *The resilience problem for the language $ab|bc|ca$ is NP-hard.*

Tractability due to submodularity. We now present our second case of tractable languages. All PTIME cases discussed in this paper, as well as those discussed in previous work on resilience for conjunctive queries [12, 13, 25], are established by reducing to MinCut problems in a flow graph constructed in linear-time data complexity. But we now show that, in our setting, some languages are tractable for a different reason, by reducing to submodular function minimization. This implies a PTIME algorithm [26], albeit with a worse polynomial degree than when reducing to MinCut.

Proposition 7.7 (Tractability via submodularity)**.** *Let $\alpha = a_1 \cdots a_n$ be a word where all letters are distinct, and let a_{n+1} be a letter not occurring in α. The problem $\mathsf{RES}_{\mathrm{bag}}(\alpha|a_{n-1}a_{n+1})$ is in PTIME.*

Hence, the query $abc|be$, or the query $abcd|ce$ from Figure 1, enjoy tractable resilience.

Proof sketch. We define a function on sets Y of domain elements whose value is the resilience when we commit to removing precisely the a_{n+1}-facts whose tail is in Y. We show that this function is submodular, relying on a known result on the modularity of maximal flows [27, Lemma 3.2(ii)], which implies that we can find the set Y minimizing the function and compute the resilience. □

Additional NP-hard cases and open problems. Already in the case of finite reduced languages, there are gaps remaining between the tractable cases (local languages, and languages covered by Proposition 7.5 or Proposition 7.7) and the hard cases (languages containing a word with a repeated letter, and four-legged languages) that we have identified so far. We show below that some of these unclassified languages are NP-hard by building specific gadgets:

Proposition 7.8. $\mathsf{RES}_{\mathrm{set}}(abcd|be|ef)$ *and* $\mathsf{RES}_{\mathrm{set}}(abcd|bef)$ *are NP-hard.*

We leave the question open for other languages, including $abcd|be$, $abc|bcd$, or $abc|bef$. Many infinite languages also remain unclassified, such as $ax^*b|xd$ for instance.

8 Conclusion and Future Work

We have studied the complexity of resilience for RPQs, towards the goal of characterizing the languages for which the problem is tractable. We study bag and set semantics, but our results show no difference between the two settings, unlike for CQs [25]. We leave open the question of whether there are languages L, regular or not, for which $\mathsf{RES}_{\mathrm{set}}(L)$ is PTIME but $\mathsf{RES}_{\mathrm{bag}}(L)$ is NP-hard. We also leave open the question of classifying the complexity of resilience in intermediate settings, e.g., bag semantics where only weights 1, and ∞ are allowed; or the setting where some relations are indicated as *exogenous* in the signature, amounting to giving a weight of ∞ to all their facts.

Our work does not fully characterize the tractability boundary and leaves some cases open. At a high level, we still do not understand the non-local languages that are not four-legged, i.e., non-letter-Cartesian languages featuring counterexamples where some of the "legs" are empty. We also do not know if the submodularity approach is needed for queries like $abc|be$, or whether tractability can be obtained in easier ways.[6]

Another open question concerns *non-Boolean RPQs*, i.e., where endpoints are fixed (or given as input). For instance, the RPQ aa then seems tractable as possible contingency sets have a simple structure; but longer RPQs may stay hard. This problem may relate to *length-bounded cuts* [5, 19], or multicuts [14] and skew multicuts [24]. Note that this non-Boolean setting corresponds to the problem of *deletion propagation with source side effects* [9], which was studied before resilience: this problem asks about finding a minimum contingency set to remove a specific query answer. The case of removing multiple tuples has also been investigated [21]. We do not know whether a dichotomy in one of these settings implies a dichotomy for our problem, or vice-versa. We also do not understand how our results relate to VCSPs [8], e.g., whether classes of VCSPs with better algorithms (in combined complexity, or in terms of the polynomial degree) can imply tractable algorithms for some RPQ classes; or whether VCSP dichotomies hold in set semantics (or in bag semantics with weights restricted to $1, \infty$).

Last, resilience can be studied for recursive queries beyond RPQs, in particular two-way RPQs or Datalog, though this would require new techniques (unlike RPQs, these queries are not "directional").

[6] After we had submitted this paper for review, an argument was pointed out by Martín Muñoz [18] implying that Proposition 7.7 can be shown using a flow encoding instead of submodular function minimization. Further, his argument seems to imply that the tractability of resilience extends to more queries, such as $abcd|be$. We leave the exploration of these consequences to a future version of the paper.

9 Acknowledgments

The authors are grateful to Charles Paperman for help about local languages, and to the reviewers for their valuable feedback. Part of this work was conducted while the authors were visiting the Simons Institute for the Theory of Computing. Amarilli was partially supported by the ANR project EQUUS ANR-19-CE48-0019, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758, and by the ANR project ANR-18-CE23-0003-02 ("CQFD"). Gatterbauer and Makhija were partially supported by the National Science Foundation (NSF) under award number IIS-1762268.

References

[1] Ravindra K Ahuja, Murali Kodialam, Ajay K Mishra, and James B Orlin. Computational investigations of maximum flow algorithms. *European Journal of Operational Research*, 97(3):509–542, 1997. doi:10.1016/S0377-2217(96)00269-X.

[2] Antoine Amarilli, Wolfgang Gatterbauer, Neha Makhija, and Mikaël Monet. Resilience for regular path queries: Towards a complexity classification. *arXiv preprint arXiv:2412.09411*, 2024.

[3] Antoine Amarilli, Wolfgang Gatterbauer, Neha Makhija, and Mikaël Monet. Code repository for gadget verification, 2024. https://osf.io/6kfj3/?view_only=b894b3ce8f0540f39233d761087ed92d.

[4] Antoine Amarilli, Timothy van Bremen, Octave Gaspard, and Kuldeep S. Meel. Approximating queries on probabilistic graphs, 2024. arXiv:2309.13287.

[5] Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Petr Kolman, Ondřej Pangrác, Heiko Schilling, and Martin Skutella. Length-bounded cuts and flows. *ACM Trans. Algorithms*, 7(1):1–27, 2010. doi:10.1145/1868237.1868241.

[6] Pablo Barceló, Diego Figueira, and Miguel Romero. Boundedness of conjunctive regular path queries. In *ICALP*, volume 132, pages 104:1–104:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.104.

[7] David A Mix Barrington, Neil Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien. First-order expressibility of languages with neutral letters or: The Crane Beach conjecture. *Journal of Computer and System Sciences*, 70(2):101–127, 2005. doi:10.1016/j.jcss.2004.07.004.

[8] Manuel Bodirsky, Žaneta Semanišinová, and Carsten Lutz. The complexity of resilience problems via valued constraint satisfaction problems. In *LICS*, pages 14:1–14:14, 2024. doi:10.1145/3661814.3662071.

[9] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. On propagation of deletions and annotations through views. In *PODS*, 2002.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, Mass., 3rd ed edition, 2009. URL: https://dl.acm.org/doi/book/10.5555/1614191.

[11] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956. doi:10.4153/cjm-1956-045-5.

[12] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. The complexity of resilience and responsibility for self-join-free conjunctive queries. *PVLDB*, 9(3):180–191, 2015. doi:10.14778/2850583.2850592.

[13] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. New results for the complexity of resilience for binary conjunctive queries with self-joins. In *PODS*, pages 271–284, 2020. doi:10.1145/3375395.3387647.

[14] Naveen Garg, Vijay V Vazirani, and Mihalis Yannakakis. Multiway cuts in directed and node weighted graphs. In *ICALP*, pages 487–498. Springer, 1994. doi:10.1007/3-540-58201-0_92.

[15] Georg Gottlob. Hypergraph transversals. In *FoIKS*, pages 1–5. Springer, 2004. doi:10.1007/978-3-540-24627-5_1.

[16] Monika Henzinger, Jason Li, Satish Rao, and Di Wang. Deterministic near-linear time minimum cut in weighted graphs. In *SODA*, pages 3089–3139. SIAM, 2024. doi:10.1137/1.9781611977912.111.

[17] Hiroshi Hirai and Ryuhei Mizutani. Minimum 0-extension problems on directed metrics. *Discrete Optimization*, 40:100642, 2021. doi:10.1016/j.disopt.2021.100642.

[18] M.Monet (https://cstheory.stackexchange.com/users/38111/m monet). Is this variant of min-cut ptime? Theoretical Computer Science Stack Exchange. URL:https://cstheory.stackexchange.com/q/54790 (version: 2024-10-29). URL: https://cstheory.stackexchange.com/q/54790, arXiv:https://cstheory.stackexchange.com/q/54790.

[19] Alon Itai, Yehoshua Perl, and Yossi Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12(3):277–286, 1982. doi:10.1002/net.3230120306.

[20] Richard M Karp. Reductibility among combinatorial problems. In *Complexity of Computer Computations. The IBM Research Symposia Series*, pages 85–103. Springer, 1972. doi:10.1007/978-1-4684-2001-2_9.

[21] Benny Kimelfeld, Jan Vondrák, and David P Woodruff. Multi-tuple deletion propagation: Approximations and complexity. *PVLDB*, 6(13), 2013.

[22] Vladimir Kolmogorov. Testing the complexity of a valued CSP language. In *ICALP*, volume 132, pages 77:1–77:12. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.77.
[23] Michal Kouckỳ, Pavel Pudlák, and Denis Thérien. Bounded-depth circuits: Separating wires from gates. In *STOC*, pages 257–265, 2005. doi:10.1145/1060590.1060629.
[24] Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 29(1):122–144, 2015. doi:10.1137/120904202.
[25] Neha Makhija and Wolfgang Gatterbauer. A unified approach for resilience and causal responsibility with Integer Linear Programming (ILP) and LP relaxations. *Proc. ACM Manag. Data*, 1(4):228:1 – 228:27, 2023. doi:10.1145/3626715.
[26] S. Thomas McCormick. Submodular function minimization. In *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages 321–391. Elsevier, 2005. doi:10.1016/S0927-0507(05)12007-6.
[27] Nimrod Megiddo. Optimal flows in networks with multiple sources and sinks. *Mathematical Programming*, 7:97–107, 1974. doi:10.1007/BF01585506.
[28] Karl Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927. doi:10.4064/fm-10-1-96-115.
[29] Jean-Éric Pin. Mathematical foundations of automata theory. https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf, 2022.
[30] Jacques Sakarovitch. *Elements of automata theory*. Cambridge University Press, 2009. doi:10.1017/CBO9781139195218.

Received December 2024; revised February 2025; accepted March 2025