Containment of UC2RPQ: the hard and easy cases

Diego Figueira

Containment of UC2RPQ: the hard and easy cases

Diego Figueira
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France

Abstract

We study the containment problem for UC2RPQ, that is, two-way Regular Path Queries, closed under conjunction, projection and union. We show a dichotomy property between PSPACE-c and EXPSPACE-c based on a property on the underlying graph of queries. We show that for any class $\mathscr{C}$ of graphs, the containment problem for queries whose underlying graph is in $\mathscr{C}$ is in PSPACE if and only if $\mathscr{C}$ has bounded *bridgewidth*. Bridgewidth is a graph measure we introduce to this end, defined as the maximum size of a minimal edge separator of a graph.

2012 ACM Subject Classification Information systems → Graph-based database models; Information systems → Resource Description Framework (RDF); Mathematics of computing → Graph theory; Theory of computation → Formal languages and automata theory

Keywords and phrases Regular Path Queries (RPQ), 2RPQ, CRPQ, C2RPQ, UC2RPQ, graph databases, containment, inclusion, equivalence, dichotomy, graph measure, bridge-width (bridgewidth), minimal edge separator, minimal cut-set, max-cut, tree-width (treewidth)

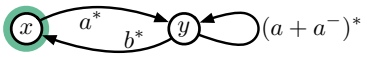
Funding ANR DéLTA (grant ANR-16-CE40-0007) and ANR QUID (grant ANR-18-CE40-0031)

Acknowledgements Thanks to Guillaume Lagarde, Matthias Niewerth and anonymous reviewers for helpful comments.

1 Introduction

Graph databases is a prominent area of study within database theory, in which the use of recursive queries is crucial [4, 2]. A graph database is a finite edge-labeled directed graph. The most basic navigational querying mechanism for graph databases corresponds to the class of *regular path queries* (RPQs), which test whether two nodes of the graph are connected by a path whose label belongs to a given regular language. RPQs are often extended with the ability to traverse edges in both directions, giving rise to the class of *two-way* RPQs, or 2RPQs [9]. For example, an regular expression like $(a + a^-)^*$ states that there is a path that can traverse a-labelled edges either in the forward or reverse direction. The core of the most popular recursive query languages for graph databases is defined by *conjunctive* 2RPQs, or C2RPQs, which are the closure of 2RPQs under conjunction and existential quantifications [7]. As an example, a C2RPQ could be described as a formula γ like

$$\gamma(x) = (\exists y)\ \ x \xrightarrow{a^*} y \ \ \wedge \ \ y \xrightarrow{(a+a^-)^*} y \ \ \wedge \ \ y \xrightarrow{b^*} x.$$



Here γ is a unary formula that outputs all vertices v_x of the graph database from which there is an a labelled path to some vertex v_y contained in some undirected a-cycle, and so that there is a directed b-path from v_y to v_x. Note that the query can also be depicted as a graph (on the right), edge-labelled with languages and with some highlighted vertices representing free (output) variables. We also consider *unions* of C2RPQs, or UC2RPQs.

The *containment problem* is arguably the most basic static analysis problem on monotone query languages. In our case it is the problem of, given two UC2RPQ γ, γ', whether $\gamma(\mathcal{G}) \subseteq \gamma'(\mathcal{G})$ for all graph databases $\mathcal{G}$.

The "four Italians" [7] have long ago shown that the containment problem for UC2RPQ is decidable, EXPSPACE-complete (in particular generalizing a prior EXPSPACE upper bound for CRPQ [11]). On the other hand, Barceló *et al.* [6] have shown that on the class $\mathscr{A}$ of

acyclic multi-graphs[1], there is a measure of '*width*' so that for every acyclic class $\mathscr{C} \subseteq \mathscr{A}$ of bounded *width*, the containment problem on UC2RPQ whose underlying graph is in $\mathscr{C}$ is in PSpace.

Contribution. Here we lift their notion of *width* to arbitrary multi-graphs. We call our measure '*bridgewidth*' and we obtain that for every class $\mathscr{C}$ of multi-graphs:

- If $\mathscr{C}$ has bounded bridgewidth, the containment problem for UC2RPQ whose underlying graph is in $\mathscr{C}$ is in PSpace;
- If $\mathscr{C}$ has unbounded bridgewidth, the containment problem for UC2RPQ whose underlying graph is in $\mathscr{C}$ is ExpSpace-complete.

Further, the lower bound hold also if we replace UC2RPQ with CRPQ or even with Boolean CRPQ, and the upper bounds hold also if we allow any arbitrary UC2RPQ in the left-hand side of the containment problem.

But what is *bridgewidth*? It is a rather intuitive measure of graphs. A *bridge* is a minimal edge separator, that is, a minimal set of edges whose removal increases the number of connected components of the graph. The bridgewidth of a graph is the maximum size of a bridge therein. As it turns out, on the class of acyclic graphs $\mathscr{A}$, bridgewidth and [6]'s *width* measure coincide.

Another related static analysis problem is the *boundedness problem* (*i.e.*, whether a UC2RPQ is equivalent to one whose languages are all finite). We have recently shown that, similarly to the containment problem, this problem is ExpSpace-complete and that on acyclic UCRPQ queries of bounded width it is in PSpace [5]. We conjecture that an adaptation of bridgewidth may also characterize the complexity for this problem for UCRPQ (see Section 7).

Organization. We start with necessary basic definitions of graphs, bridges, graph databases, automata, and UC2RPQ in Section 2, and we formally state the main result in Section 3. Section 4 contains some technical lemmas needed for the upper bound algorithm. Finally, Sections 5 and 6 spell out the details of the upper and lower bounds of the main theorem, respectively. We conclude with some remarks in Section 7.

2 Preliminaries

Graphs

A **multi-graph** $M = (V, E, \eta)$ is a finite set of vertices V and edges E together with a function $\eta : E \to V \times V$ associating each edge with the source and target vertices. For convenience we will sometimes use $\eta_1(e)$ and $\eta_2(e)$ to denote the vertex in the first and second components of $\eta(e)$. For economy, we will henceforth write 'graph' to denote a multi-graph. For any set of edges $E' \subseteq E$ and set of vertices $V' \subseteq V$, we write $E'[V']$ to denote the set of all edges of E' **incident** to V', that is $E'[V'] = \{e \in E' : \eta_i(e) \in V' \text{ for some } i \in \{1, 2\}\}$; for a vertex $v \in V$, we write $E'[v]$ as short for $E'[\{v\}]$. A **connected component** of M is a non-empty minimal set of vertices $V' \subseteq V$ so that $\eta(e) \in (V' \times V') \dot\cup ((V \setminus V') \times (V \setminus V'))$ for every $e \in E$. A graph is **connected** if it has only one connected component. Henceforward, whenever we write 'minimal' it is with respect to set-containment. An **isomorphism** between graphs $M = (V, E, \eta)$ and $M' = (V', E', \eta')$, noted $M \cong M'$, is a pair of bijections $\nu_V : V \to V'$ and $\nu_E : E \to E'$ such that $\eta(e) = (u, v)$ if and only if $\eta'(\nu_E(e)) = (\nu_V(u), \nu_V(v))$. We

[1] In this context, by acyclic we mean any directed multi-graph such that every cycle of its underlying undirected simple graph is a self-loop.

will henceforth always work modulo graph isomorphism. Given a set $W \subseteq V$, the **graph induced** by W, written $M|_W$, is $\langle W, \{e \in E : \eta(e) \in W \times W\}, \{e \mapsto \eta(e) : \eta(e) \in W \times W\}\rangle$. Similarly, for $E' \subseteq E$, $M|_{E'} = \langle V, E', \{e \mapsto \eta(e) : e \in E'\}\rangle$. Given an equivalence relation $\sim \subseteq V \times V$, the **quotient graph** $M/\sim$ is defined as (V', E, η') where V' has one vertex $[v]_\sim$ for every $\sim$ equivalence class, and $\eta'(e) = ([v]_\sim, [v']_\sim)$ for every $e \in E$ such that $\eta(e) = (v, v')$.

Bridges

A **bridge** (*a.k.a.* minimal edge separator) of a graph M is a minimal set $B \subseteq E$ of edges (minimal in the sense of set-containment) whose deletion induces an increase on the number of connected components of M. The **bridgewidth** of M, which we note $bw(M)$, is the

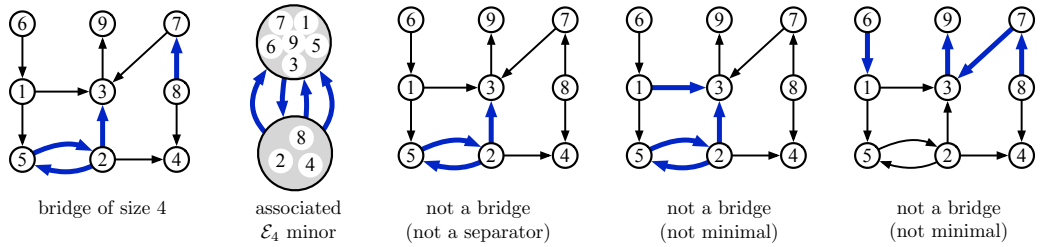


Figure 1 Examples of bridges.

maximum size of a bridge of M, or 0 if M has no edges. The bridgewidth of a class of graphs $\mathscr{C}$, which we write $bw(\mathscr{C})$, is defined as $\sup_{M \in \mathscr{C}} bw(M)$. If $bw(\mathscr{C}) < \infty$, we say that $\mathscr{C}$ has **bounded bridgewidth**; otherwise, it has **unbounded bridgewidth**.

Two-way alternating automata

The upper bound algorithm makes use of automata, we give here our *sui generis* definition of 2AFA: a simplified weaker version of the usual 2AFA which fulfills our needs. A **2-way alternating finite state automaton (2AFA)** is a tuple $\mathcal{A} = \langle \mathbb{A}, Q, I, F, \delta\rangle$ where $\mathbb{A}$ is a finite alphabet of letters; Q is a finite set of states; $I, F \subseteq Q$ are sets of initial and final states respectively; and $\delta : Q \to \mathcal{B}_\vee(\mathcal{B}_\wedge(\{+1, -1\} \times \mathbb{A} \times Q))$ is the transition function, where $\mathcal{B}_\vee(\mathcal{B}_\wedge(X))$ stands for a disjunction of conjunction of elements from X (*e.g.*, "$(+1, a, q) \vee ((+1, a, p) \wedge (-1, b, p)) \vee (-1, a, p)$"). A **non-deterministic two-way finite automaton (2NFA)** is a 2AFA that has no conjunctions '$\wedge$' in δ, that is, $\delta : Q \to \mathcal{B}_\vee(\{+1, -1\} \times \mathbb{A} \times Q)$. If further δ has no '-1' elements (*i.e.*, if $\delta : Q \to \mathcal{B}_\vee(\{+1\} \times \mathbb{A} \times Q)$), it is a **non-deterministic finite automaton (NFA)**. An **run from position** i on a word $w \in \mathbb{A}^*$ (where $0 \leq i \leq |w|$) is a finite non-empty tree whose vertices are labelled by elements from $\{0, \ldots, |w|\} \times Q$ such that the root is labeled by some element from $\{i\} \times I$, and for every vertex x labelled (j, q) there is a disjunct of $\delta(q)$ such that for every conjunct (n, a, p) thereof we have: (i) $0 \leq j + n \leq |w|$, (ii) $(n, a) \in \{(+1, w[j+1]), (-1, w[j])\}$, and (iii) there is a child of x labelled $(j + n, p)$. If the automaton is an NFA, note that the run has only one leaf (j, q_f), in which case we say that the run starts at position i and *ends* at position j. A run is **accepting** if every leaf is labeled by an element from $\{|w|\} \times F$. In the sequel it will be convenient to use these automata as *unary querying devices* on finite words; that is, for any given word $w \in \mathbb{A}^*$, the evaluation $\mathcal{A}(w)$ of the automaton on the word, outputs the set of positions $\mathcal{A}(w) \subseteq \{0, \ldots, |w|\}$ of w from which there is an accepting run. The **language** of $\mathcal{A}$ is the set of all words w such that $0 \in \mathcal{A}(w)$. For a word w, we will write $w \in \mathcal{A}$ to denote

that w is in the language recognized by $\mathcal{A}$. For any given $\delta : Q \to \mathcal{B}_\vee(\mathcal{B}_\wedge(\{+1,-1\} \times \mathbb{A} \times Q))$ and $I, F \subseteq Q$ we denote by $\delta[I,F]$ the 2AFA having I and F as initial and final state sets and δ as transition function (the alphabet and statespace being implicit).

Oftentimes 2-way automata are defined to have also "word delimiters", in order to recognize when we are at the leftmost or rightmost position of the word. We do not need this feature in our construction (and its absence simplifies, albeit slightly, some definitions); we therefore prefer to leave the definition as simple as possible.

Graph databases and UC2RPQ

A graph database is an edge-labelled finite graph, where labels come from some fix, finite alphabet $\mathbb{A}$. Formally, a **graph database** over an alphabet $\mathbb{A}$ (henceforth graph db) $\mathcal{G} = \langle V, E, \eta, \lambda \rangle$ is a graph $\langle V, E, \eta \rangle$ equipped with a function $\lambda : E \to \mathbb{A}$.

We work with query languages that can traverse edges in both directions: in the direction of the edge as represented in the graph db (*i.e.*, in the *forward* direction), or in the opposite direction (*i.e.*, in the *reverse* direction). Given a finite alphabet $\mathbb{A}$ we represent the instruction of traversing an a-labelled edge in the forward direction by reading the letter $a \in \mathbb{A}$, and the instruction of traversing an a-labelled edge in the reverse direction by reading a^-. Hence, let $\mathbb{A}^-$ be the set of all letters a^- where $a \in \mathbb{A}$, and let $\mathbb{A}^\pm$ be $\mathbb{A} \dot\cup \mathbb{A}^-$. For every $a \in \mathbb{A}$, let $(a^-)^\neg = a$ and $a^\neg = a^-$. A C2RPQ is the closure under conjunction and existential quantification of 2RPQ queries, which are of the form $L(x,y)$ where L is any regular language over $\mathbb{A}^\pm$ and x, y are free variables ranging over vertices of graph databases. Here, we prefer to define C2RPQ directly in a graph form. Let us first define informally what a C2RPQ is —we will later deal with all boring details. A C2RPQ γ is a graph whose edges are labelled with regular languages over $\mathbb{A}^\pm$, equipped with a vector of 'output' vertices. An expansion of a C2RPQ is the result of replacing every edge from x to y labelled by L with a path corresponding to some word $w \in L$, respecting the directions imposed by the alphabet $\mathbb{A}^\pm$. As a result, we obtain a graph db and a vector of vertices. For example if $w = a \cdot b^- \cdot b$, then the path looks like $x \xrightarrow{a} \bullet \xleftarrow{b} \bullet \xrightarrow{b} y$; and if $w = \varepsilon$ then the path is empty, and it forces the collapse of x and y. Some examples are shown in Figure 2. Observe that each expansion is

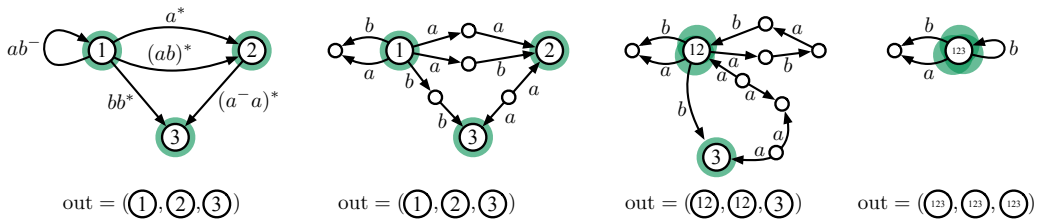


Figure 2 A C2RPQ (left) and three possible expansions. Highlighted vertices are output vertices.

essentially a Conjunctive Query (CQ). A tuple of vertices of a graph db $\mathcal{G}$ is in the output $\gamma(\mathcal{G})$ of a UC2RPQ γ evaluated at $\mathcal{G}$ iff it is in the output $Q(\mathcal{G})$ of the CQ Q corresponding to some expansion of γ.

Formal details follow. A **C2RPQ** over the alphabet $\mathbb{A}$ is represented as

$$\gamma = \langle V, E, \eta, Q, \delta, I, F, \bar{o} \rangle$$

where

(i) $\langle V, E, \eta \rangle$ is a graph,

(ii) Q is a finite set of states,
(iii) $\delta : Q \to \mathcal{B}_\vee(\{+1\} \times \mathbb{A}^\pm \times Q)$ is the transition relation of an NFA over $\mathbb{A}^\pm$,
(iv) $I, F : E \to 2^Q$ are functions indicating the set of initial and final state for each edge, and
(v) $\bar{o}$ is a (possibly null-ary) vector over V, called the **output vector**.

We henceforth write $\bar{\emptyset}$ to denote the null-ary vector. We often write $\gamma(\bar{o})$, whenever $\bar{o} \neq \bar{\emptyset}$, to make explicit the output vector of γ. If $\bar{o} = \bar{\emptyset}$, we say that γ is a **Boolean** C2RPQ. The **arity** of γ is the dimension of $\bar{o}$ (hence, 0 if $\bar{o} = \bar{\emptyset}$). The **underlying graph** of γ is $\langle V, E, \eta \rangle$. We denote by $|\gamma|$ the size of any reasonable representation of γ.

For an equivalence relation $\sim \subseteq V \times V$, the quotient $(\mathcal{G}, \bar{o})/\sim$ of a C2RPQ is defined as for graphs, that is, the result of replacing every vertex v by a representative element $[v]_\sim$ from its equivalence class.

For every word $w \in (\mathbb{A}^\pm)^*$ of length n, let $w[i]$ denote its i-th letter. The graph db associated to w contains $n+1$ distinct vertices $v_0, \ldots, v_n$ and n edges $e_1, \ldots, e_n$ so that for every i: if $w[i] \in \mathbb{A}$, then $\lambda(e_i) = w[i]$ and $\eta(e_i) = (v_{i-1}, v_i)$; if $w[i] \in \mathbb{A}^-$, then $\lambda(e_i) = (w[i])^\neg$ and $\eta(e_i) = (v_i, v_{i-1})$. We call this graph db the **semipath for the word w that starts in v_0 and ends in v_n**. We will also refer to the **i-th vertex of the semipath for w** to denote the vertex v_i. Note that if $w = \varepsilon$ then $v_0 = v_n$, and in this case we say that the semipath is **empty**. By $\vec{\delta}$ we denote transition function of a 2-way NFA having Q as statespace, defined as $q \mapsto \bigvee\{(+1, \alpha, q'), (-1, \alpha^\neg, q') : (+1, \alpha, q') \text{ in } \delta(q)\}$. The intuition is that $\vec{\delta}$ 'implements' the notion that by reading a^- we traverse a-edges in the *reverse* direction. More concretely, for a semipath $\mathcal{G}_w$ for $w \in (\mathbb{A}^\pm)^*$, consider $\mathcal{P}_w$ the graph db obtained by adding to $\mathcal{G}_w$ an edge labelled a^- from u to v for every edge labelled a from v to u. Observe then that a run of $\vec{\delta}$ between states p and q on w corresponds to a run of δ between p and q on a directed path of $\mathcal{P}_w$, and vice-versa. A **semipath for an edge e** of a C2RPQ as above is any non-empty semipath for a word $w \in \delta[I(e), F(e)]$ which starts in $\eta_1(e)$ and ends in $\eta_2(e)$ (note that, in particular, $w \neq \varepsilon$ and $\eta_1(e) \neq \eta_2(e)$).

$(\mathcal{X}, \bar{x})$ is an **expansion** of $\gamma(\bar{o})$ (*a.k.a.* a canonical database for $\gamma(\bar{o})$) if $\mathcal{X}$ is a graph db over $\mathbb{A}$, $\bar{x}$ is a vector of vertices of $\mathcal{X}$ of the same arity as $\bar{o}$, and there exists a partition of E into E_0, E_1 (*i.e.*, $E = E_0 \dot\cup E_1$) so that

- for each $e \in E_0$, $I(e) \cap F(e) \neq \emptyset$; and
- for each $e \in E_1$, there is a semipath π_e for e,

and $\mathcal{X}$ is the union of all these semipaths, collapsing all pairs of vertices of E_0. Formally, $(\mathcal{X}, \bar{x})$ is defined as $((\bigcup_{e \in E_1} \pi_e), \bar{o})/\sim$, where $\sim$ is the equivalence relation induced by the connected components of $\mathcal{X}|_{E_0}$.

A **UC2RPQ** is a finite union $\gamma = \gamma_1 \cup \cdots \cup \gamma_n$ of C2RPQ with the same arity. The set of expansions of γ is the union of the sets of expansions of the γ_i's.

A vector $\bar{v}$ of vertices of a graph db $\mathcal{G}$ is in the output $\gamma(\mathcal{G})$ of UC2RPQ γ evaluated on $\mathcal{G}$ iff there is some expansion $(\mathcal{X}, \bar{x})$ of γ and a homomorphism $h : \mathcal{X} \to \mathcal{G}$ such that $h(\bar{x}) = \bar{v}$. If γ is Boolean, we say that it is *true* in $\mathcal{G}$ iff $\bar{\emptyset} \in \gamma(\mathcal{G})$, that is, if there exists a homomorphism $\mathcal{X} \to \mathcal{G}$ for some expansion $\mathcal{X}$.

The bridgewidth of a UC2RPQ is the maximum bridgewidth of its underlying graphs, and the bridgewidth of a graph db is the bridgewidth of its underlying graph.

Containment problem for query fragments

Given two UC2RPQ γ_1, γ_2 we say that γ_1 **is contained in** γ_2 (and we note it $\gamma_1 \subseteq \gamma_2$) if γ_1 and γ_2 have the same arity (possibly 0) and $\gamma_1(\mathcal{G}) \subseteq \gamma_2(\mathcal{G})$ for all graph databases $\mathcal{G}$. For any fragment $\mathcal{F}$ of UC2RPQ, the **$\mathcal{F}$-containment problem** is the problem of deciding,

given $\gamma_1, \gamma_2 \in \mathcal{F}$, whether $\gamma_1 \subseteq \gamma_2$. Given a class of graphs $\mathscr{C}$, the query class CRPQ($\mathscr{C}$) [resp. C2RPQ($\mathscr{C}$), UC2RPQ($\mathscr{C}$)] is the set of all CRPQ [resp. C2RPQ, UC2RPQ] whose underlying graph is in $\mathscr{C}$.

3 Main result

Let us say that a class of graphs is **non-trivial**, if it contains at least one graph with at least one edge. The main result is the following.

▶ **Theorem 1** (containment dichotomy). *For every non-trivial class $\mathscr{C}$ of graphs,*

1. *if $\mathscr{C}$ has bounded bridgewidth, the UC2RPQ($\mathscr{C}$)-containment problem is* PSpace*-complete;*
2. *if $\mathscr{C}$ has unbounded bridgewidth, the UC2RPQ($\mathscr{C}$)-containment problem is* ExpSpace*-complete.*

Further, the lower bounds hold also for the fragment of Boolean CRPQ($\mathscr{C}$).

Proofs for lower and upper bounds build upon known results and techniques in the area. Although technical details are somewhat lengthy, they do not bring original ideas other than verifying that bridgewidth is the 'right' notion for a dichotomy result. In particular, Barceló *et al.* [6] have already shown the result for the restricted case where $\mathscr{C}$ is any class of 'acyclic' graphs (meaning that the only allowed cycles in the underlying undirected simple graphs are self-loops). They use a notion of '*width*' which coincides with bridgewidth on this class of graphs. The lifting of this result to bridgewidth is considerably more involved, but it follows the same philosophy.

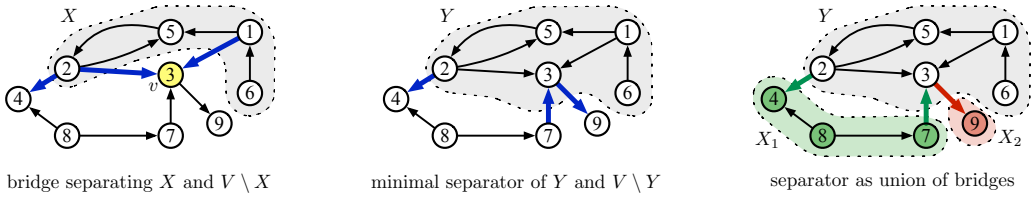
4 A bridge maintenance toolkit

We state here some properties of bridges and definitions which shall be of use in the next section.

For a graph $M = \langle V, E, \eta \rangle$, we say that a set of edges $S \subseteq E$ **separates** $X \subseteq V$ and $Y \subseteq V$ if all elements of $X \cup Y$ are in the same connected component of M, and for every $x \in X$ and $y \in Y$, x and y are in different connected components of $M|_{E \setminus S}$. In this case we say that S is a **separator** of X and Y; and if it is minimal with respect to this property, we say that it is a **minimal separator**. Observe that a set of edges is a bridge if and only if it is a minimal separator of two singleton sets $\{x\}, \{y\}$. However, minimal separators need not be bridges in general; for example, the rightmost picture in Figure 1 (page 3) is a minimal separator of $\{①, ③, ⑧\}$ and $\{⑥, ⑨, ⑦\}$.

For a bridge $B \subseteq E$ we say that $X \subseteq V$ is a **side** of B if $X \neq \emptyset$ and there is a connected component Y of $M \setminus B$ such that $X = \{y \in Y : B[y] \neq \emptyset\}$. Note that there are exactly two sides for each bridge, and every bridge separates its sides. For example, the bridge in the leftmost picture of Figure 1 has $\{⑤, ③, ⑦\}$ and $\{②, ⑧\}$ as sides. For some set of vertices $Z \subseteq V$ we say **the Z-side of B** to denote the side of B that intersects Z (assuming there is exactly one).

Bridgewidth can be also understood in terms of graph minors, as we show next. Given an edge $e \in E$ of $M = (V, E, \eta)$ with $\eta(e) = (u, v)$ for some $u \neq v$, an **e-edge contraction** of M is the graph M' obtained by collapsing the endpoints of e. Formally, $M' = (M|_{E \setminus \{e\}})/\sim$ for $\sim$ the finest equivalence relation such that $u \sim v$ (*i.e.*, $\sim = \{(z, z) : z \in V\} \cup \{(u, v), (v, u)\}$). A **minor** of M is any graph M' obtained from M by contracting edges and removing edges and vertices. In particular, minors preserve boundedness of bridgewidth, so do subdivisions and, as a consequence, so do expansions (since graphs corresponding to expansions are subdivisions of minors).



bridge separating X and $V \setminus X$ | minimal separator of Y and $V \setminus Y$ | separator as union of bridges

Figure 3 Illustration for Lemma 5 (first two pictures) and Lemma 4 (last two pictures).

▶ **Lemma 2.**
- *For every graph M and minor M' thereof, $bw(M) \geq bw(M')$.*
- *For every UC2RPQ γ and expansion $\hat{\gamma}$ thereof, $bw(\gamma) \geq bw(\hat{\gamma})$.*

Let us call $\mathcal{E}_k$ the class of graphs containing two distinct nodes u, v and k edges between these nodes. That is, the set of all graphs $M = (\{u, v\}, \{e_1, \ldots, e_k\}, \eta)$ for any η such that $\eta(e_i) \in \{(u, v), (v, u)\}$ for every i. Note that $|\mathcal{E}_k| = 2^k$ (actually, there are just $\lceil \frac{n}{2} \rceil$ many graphs up to isomorphism). It is easy to see that the presence of some $\mathcal{E}_k$ minor witnesses a bridge of size at least k and vice-versa (see first two pictures of Figure 1).

▶ **Lemma 3.** *A graph M has bridgewidth at least k if and only if it contains some graph from $\mathcal{E}_k$ as minor.*

▶ **Lemma 4.** *If $S \subseteq E$ is a minimal separator of Y and $V \setminus Y$ in a graph $M = (V, E, \eta)$ such that $M|_Y$ is connected, then there is a partition $\{Y_i\}_{i \in I}$ of $V \setminus Y$ and a pairwise disjoint set of bridges $\{B_i\}_{i \in I}$, computable in polynomial time, so that*
- *$\bigcup_{i \in I} B_i = S$; and*
- *B_i separates Y_i and $V \setminus Y_i$ for every $i \in I$.*

Proof. Let $\{Y_i\}_{i \in I}$ be the partition of $V \setminus Y$ in the connected components of $M|_{V \setminus Y}$. For every i, let $B_i \subseteq S$ be the set of edges between Y and Y_i. It follows that every B_i is a bridge separating Y_i from $V \setminus Y_i$, no edge can belong to two distinct B_i's, and every edge of S is in some B_i. See last two pictures of Figure 3 for an example. ◀

▶ **Lemma 5.** *If $B \subseteq E$ is a bridge of a graph (V, E, η) separating X and $V \setminus X$, and $v \in V \setminus X$ is a vertex incident to B (i.e., so that $B[v] \neq \emptyset$), then there is a (possibly empty) partition $\{X_i\}_{i \in I}$ of $V \setminus (X \cup \{v\})$ and a pairwise disjoint set of bridges $\{B_i\}_{i \in I}$ such that*
- *$\bigcup_{i \in I} B_i = (B \cup E[v]) \setminus B[v]$; and*
- *B_i separates X_i and $V \setminus X_i$ for every $i \in I$.*

Proof. Consider $Y = X \cup \{v\}$ and observe that $M|_Y$ is connected. Then apply the previous Lemma 4 to the minimal separator $S = (B \cup E[v]) \setminus B[v]$ of Y and $V \setminus Y$. See first two pictures of Figure 3. ◀

▶ **Lemma 6.** *For every graph $M = (V, E, \eta)$ there is a vertex $v \in V$ so that $E[v]$ is a bridge.*

Proof. Suppose wlog that M is connected. Observe that if there is a bridge separating X and $V \setminus X$ and $|V \setminus X| = 1$ we are done: the singleton set $V \setminus X$ yields the vertex v to choose. Otherwise, we proceed by induction on the size of $V \setminus X$. Take any bridge B separating X from $V \setminus X$, and take any $v \in V \setminus X$ incident to B. By Lemma 5, $(B \cup E[v]) \setminus B[v]$ is a disjoint union of bridges. Take any such bridge B and observe that it separates X' and $V \setminus X'$ for some $X' \supseteq X \cup \{v\}$. Since $|V \setminus X'| < |V \setminus X|$, we apply inductive hypothesis and we conclude that there must be a vertex in $V \setminus X'$ verifying the property. ◀

▶ **Lemma 7.** *Given a connected graph M and a partition $\tilde{X} \dot\cup X_L \dot\cup X_R$ of the set of vertices therein, and given a bridge B of M separating $\tilde{X}$ and $X_L \dot\cup X_R$; there is a partition $\{X_i\}_{i\in I}$ of $X_L \dot\cup X_R$ and a set of bridges $\{B_i\}_{i\in I}$ of M, computable in polynomial time, such that*

- **(i)** *every B_i separates X_i and $V \setminus X_i$;*
- **(ii)** *for every i there is $Z \in \{X_L, X_R\}$ such that B_i contains only edges between Z and $V \setminus Z$;*
- **(iii)** *for every i, $B_i \cap B[X_L] = \emptyset$ if and only if $B_i \cap B[X_R] \neq \emptyset$;*
- **(iv)** *for every $e \in B$, there is exactly one $i \in I$ with $e \in B_i$.*

Proof. Consider the partition $\{Y_i\}_{i\in J}$ of $X_L \dot\cup X_R$ given by all the connected components of $M|_{X_R}$ and $M|_{X_L}$, and consider the set of sets of edges $\{E_i\}_{i\in J}$ where E_i is the set of edges between Y_i and $V \setminus Y_i$. If follows that $\{Y_i\}_{i\in J}$ and $\{E_i\}_{i\in J}$ satisfy all items above, but some E_i's may not be bridges. We show how to produce a partition into bridges from this initial partition. If there is some Y_i from which there is no edge to $\tilde{X}$ but there are edges to $Y_{i_1}, \dots, Y_{i_\ell}$, then remove all $Y_i, Y_{i_1}, \dots, Y_{i_\ell}$ from the partition and replace them with $Y = Y_i \cup Y_{i_1} \cup \cdots \cup Y_{i_\ell}$. Similarly, remove $E_i, E_{i_1}, \dots, E_{i_\ell}$ and add the set of edges between Y and $V \setminus Y$. Note that this results in a strictly coarser partition of $X_L \dot\cup X_R$ which still satisfies all items with respect to its associated set of sets of edges. Repeat this operation until all sets of the partition have at least one edge to $\tilde{X}$. It follows that for each set Z of the partition obtained, the edges between Z and $V \setminus Z$ are a bridge B of M, and that the set of all these bridges verify the conditions with respect to the partition. See Figure 4 for a picture. ◀

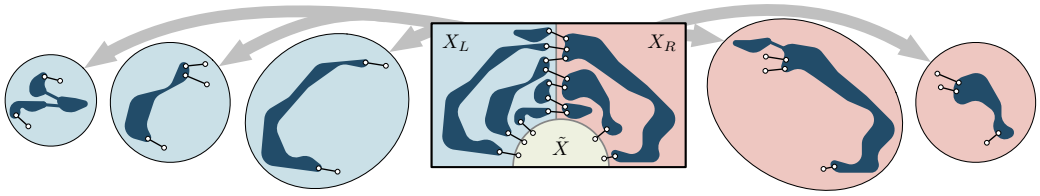


Figure 4 We amalgamate connected components that do not have incident edges with $\tilde{X}$. We end up with a partition whose every element induces a connected subgraph, and a bunch of bridges between partition elements. Each of the ovals shows a bridge and the component that it separates. Left bridges (in blue) use some edge from $B[X_L]$, and right bridges (red) use some edge from $B[X_R]$, but notice that no bridges use both an edge from $B[X_L]$ and an edge from $B[X_R]$.

5 Upper bounds

We show an algorithm to solve the containment problem for UC2RPQ which uses space exponential only in the bridgewidth of the underlying graph. Hence, if the bridgewidth is bounded, the algorithm runs in polynomial space, and otherwise in exponential space.

No self-loops assumption. To simplify some developments, we will assume that the C2RPQs we work with have no self-loops, that is, there are no edges e with $\eta_1(e) = \eta_2(e)$. Any C2RPQ can be transformed into a self-loop-less C2RPQ by adding, for every self-looping edge e on a vertex v a new vertex v_e and edge e', redefining $\eta(e) = (v, v_e)$, $\eta(e') = (v_e, v)$ and $I(e') = F(e') = F(e)$. Note that this is a linear time procedure that does not increase the bridgewidth unless the bridgewidth is 1, in which case it becomes of bridgewidth 2. Hence, this assumption is without loss of generality.

5.1 Proof strategy

We use the same proof strategy as in [6], which we review briefly here:

R1 By a result from [14, Theorem 4], there is a polynomial time reduction of this problem to the containment of a Boolean single-edge C2RPQ γ' into a Boolean UC2RPQ γ, and it is easy to see that this reduction preserves bridgewidth —in fact, $bw(\gamma') = 1$ and the underlying graph of γ is left unaltered in the cited reduction.

R2 We can reduce, in turn, this problem to the non-emptiness of a NFA $\mathcal{A}_{\gamma' \subseteq \gamma}$ of size $O(2^{|\gamma'|+|\gamma|^{c \cdot bw(\gamma)}})$, which can de done in deterministic space $O(|\gamma'| + |\gamma|^{c \cdot bw(\gamma)})$. The NFA runs over the exponential alphabet $\mathbb{A}^{\pm} \dot{\cup} \text{Loops}$, where $\text{Loops} = 2^{Q \times Q}$ and Q is the statespace of γ. This NFA is obtained from three automata $\mathcal{A}_{cd}, \mathcal{A}_{loop}, \mathcal{A}_{\gamma}$ as $\mathcal{A}_{\gamma' \subseteq \gamma} \stackrel{\text{def}}{=} \mathcal{A}_{cd} \cap \mathcal{A}_{loop} \cap \overline{\mathcal{A}_{\gamma}}$ where:

- **(i)** $\mathcal{A}_{cd}$ is a singly exponential size NFA (with a polynomial number of states) depending on γ', recognizing all words $L_0 a_1 L_1 \cdots a_n L_n \in \text{Loops} \cdot (\mathbb{A}^{\pm} \cdot \text{Loops})^*$ such that $a_1 \cdots a_n \in \delta'[I(e), F(e)]$ for e the sole edge of γ' (*i.e.*, $a_1 \cdots a_n$ represents an expansion of γ').
- **(ii)** $\mathcal{A}_{loop}$ is a singly exponential size NFA depending on γ, recognizing all words of the form $L_0 a_1 L_1 \cdots a_n L_n \in \text{Loops} \cdot (\mathbb{A}^{\pm} \cdot \text{Loops})^*$ such that, for every i, $(q, q') \in L_i$ if and only if there is a 2-way run of $\vec{\delta}[\{q\}, \{q'\}]$ on $a_1 \cdots a_n$ that starts in position i with state q and ends in the same position i with state q'.
- **(iii)** $\mathcal{A}_{\gamma}$ is a 2AFA of size $O(|\gamma|^{c \cdot bw(\gamma)})$ for some constant c with the property that for every word $w = L_0 a_1 \cdots a_n L_n \in \mathcal{A}_{loop}$, we have that $w \in \mathcal{A}_{\gamma}$ if and only if γ holds true in $\mathcal{G}_w$, where $\mathcal{G}_w$ is the semipath for $a_1 \cdots a_n$. Remember that this is equivalent to asking whether there is an expansion of γ that can be homomorphically mapped to $\mathcal{G}_w$. $\overline{\mathcal{A}_{\gamma}}$ is the automaton recognizing the complement language.

Since the complement of a 2AFA can be constructed as a NFA with a single exponential blowup in the statespace [8, Theorem 8], it follows that the resulting NFA $\mathcal{A}_{\gamma' \subseteq \gamma}$ is of size $O(2^{|\gamma'|+|\gamma|^{c \cdot bw(\gamma)}})$. Consequently, it is of single exponential size whenever the bridgewidth is bounded, and thus its emptiness can be checked using polynomial space. We invite the curious reader to read [6, §4.2] for more details on the two reductions **R1** and **R2** above.

The sole contribution of this paper on the proof above lies in the definition of $\mathcal{A}_{\gamma}$ of item **(iii)**. In [6], $\mathcal{A}_{\gamma}$ was defined for the case of γ being acyclic, and shown to be exponential in the "width of the acyclic query" (meaning the maximum number of edges between any two distinct vertices), and hence polynomial if the width is bounded. Here we lift this result to *all* queries with respect to the bridgewidth of the query, without assuming any further restriction (such as acyclicity). Bridgewidth is a generalization of their width measure, in the sense that for all acyclic queries, bridgewidth coincides with [6]'s width notion. The price to pay for this generalization is that now the definition of $\mathcal{A}_{\gamma}$ is considerably more involved. The rest of this section will be devoted to defining $\mathcal{A}_{\gamma}$ in such a way that it is exponential only in the bridgewidth of γ, and that satisfies the property described in item **(iii)**.

5.2 Definition of $\mathcal{A}_{\gamma}$

For a UC2RPQ $\gamma = \gamma_1 \cup \cdots \cup \gamma_n$, we define $\mathcal{A}_{\gamma}$ to be the union $\mathcal{A}_{\gamma_1} \cup \cdots \cup \mathcal{A}_{\gamma_n}$, this is why for simplicity we will henceforth assume that γ is a C2RPQ (*i.e.*, no unions). Let $\gamma = \langle V, E, \eta, Q, \delta, I, F, \bar{\emptyset} \rangle$ (remember, by **R1** γ is Boolean), and let $M = \langle V, E, \eta \rangle$ be the underlying graph of γ. If M is not connected —say γ is equivalent to $\gamma_1 \wedge \cdots \wedge \gamma_n$ for connected C2RPQ's— $\mathcal{A}_{\gamma}$ is defined as $\mathcal{A}_{\gamma_1} \cap \cdots \cap \mathcal{A}_{\gamma_n}$, which can be done in polynomial time since we are working with *alternating* automata. Therefore, let us also suppose, without loss of

generality, that M is connected. For any word $w = L_0 a_1 L_1 \cdots a_n L_n \in \text{Loops} \cdot (\mathbb{A}^{\pm} \cdot \text{Loops})^*$, let $\mathcal{G}_w$ denote the semipath for $a_1 \cdots a_n$.

Let us first refresh what $\mathcal{A}_\gamma$ is supposed to do. Remember that property **(iii)** concerns the case where the input is a word of the form $w = L_0 a_1 L_1 \cdots a_n L_n$ in which each L_i contains the loop information of $\vec{\delta}$ on position i of the word $a_1 \cdots a_n \in (\mathbb{A}^{\pm})^*$. Since γ is Boolean (*i.e.*, a *property* of graph db's), upon reading such a word, $\mathcal{A}_\gamma$ must check whether γ is true on $\mathcal{G}_w$, possibly using the information contained in the labels L_i's. Further, $\mathcal{A}_\gamma$ must use a 'small' set of states (polynomial if $bw(\gamma)$ is bounded).

A detour through non-Boolean queries. The definition of $\mathcal{A}_\gamma$ will make use of non-Boolean subqueries of γ. Suppose $B \subseteq E$ is a bridge separating Y from $V \setminus Y$ in M, and let $X = \{v \in Y : B[v] \neq \emptyset\}$ be the Y-side of B. For any given a state assignment $f : B \to Q$ we define the query $\gamma[B, X, f]$ as the result of modifying γ by:

- removing all edges internal to X;
- removing all vertices from $Y \setminus X$ (and the incident edges);
- defining X as output vertices[2]; and
- for every $e \in B$ redefining $I(e)$ [resp. $F(e)$] as $f(e)$ if $\eta_1(e) \in X$ [resp. if $\eta_2(e) \in X$].

See Figure 5 for an example.

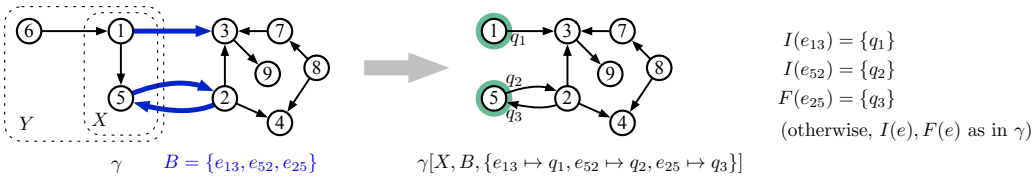


Figure 5 Highlighted vertices are output vertices and edges e adorned with a state q means that $I(e)$ is replaced with $\{q\}$ if the edge is outgoing from an output vertex, and that $F(e)$ is replaced with $\{q\}$ if e is incoming to an output vertex.

How many distinct $\gamma[B, X, f]$'s are there? The number of such queries is bounded by $(|Q| + 1)^{bw(\gamma)} \cdot 2 \cdot |E|^{bw(\gamma)}$, hence a polynomial number if the bridgewidth is bounded by a constant.

For every such non-Boolean $\gamma[B, X, f]$ we define a 2AFA automaton $\mathcal{A}_{\gamma[B,X,f]}$ with the property that, for every $0 \leq i \leq n$ and $w \in \mathcal{A}_{loop}$,

$$2i \in \mathcal{A}_{\gamma[B,X,f]}(w) \quad \text{if and only if} \quad \underbrace{(v_i, \ldots, v_i)}_{|X| \text{ times}} \in \gamma[B, X, f](\mathcal{G}_w), \tag{†}$$

where v_i is the i-th vertex of $\mathcal{G}_w$. That is, the automaton $\mathcal{A}_{\gamma[B,X,f]}$ checks whether there is an expansion of $\gamma[B, X, f]$ that can map to $\mathcal{G}_w$ via a homomorphism that assigns the i-th vertex of $\mathcal{G}_w$ to every vertex of X.

Once we know how to define the $\mathcal{A}_{\gamma[B,X,f]}$'s, the definition of $\mathcal{A}_\gamma$ follows easily: Choose any vertex v of γ such that $E[v]$ is a bridge (it exists due to Lemma 6), and guess a function $f : E[v] \to Q$ such that $f(e) \in I(e)$ if $\eta_1(e) = v$, and $f(e) \in F(e)$ if $\eta_2(e) = v$. Then, move non-deterministically to an even position, and run $\mathcal{A}_{\gamma[E[x],\{x\},f]}$. Note that, assuming $\mathcal{A}_{\gamma[E[x],\{x\},f]}$ satisfies (†), property **(iii)** holds on a word w if and only if there is an even position $2i$ of w and a function f with the aforementioned properties such that $2i \in \mathcal{A}_{\gamma[E[x],\{x\},f]}(w)$.

[2] That is, defining any vector $(v_1, \ldots, v_n)$ as output if $X = \{v_1, \ldots, v_n\}$, the order is inconsequential.

Why do we want to define $\mathcal{A}_\gamma$ in terms of $\mathcal{A}_{\gamma[B,X,f]}$? Because in this way $\mathcal{A}_{\gamma[B,X,f]}$ can be defined in an recursive way: each $\mathcal{A}_{\gamma[B,X,f]}$ is defined using other $\mathcal{A}_{\gamma[B',X',f']}$'s; which is arguably simpler to define and understand, and it has an explicit invariant.

5.3 Definition of $\mathcal{A}_{\gamma[B,X,f]}$

We will show how to construct $\mathcal{A}_{\gamma[B,X,f]}$ satisfying property (†) by possibly 'calling', as subroutines, other automata $\mathcal{A}_{\gamma[B',X',f']}$'s. Of course, we adopt this way of defining $\mathcal{A}_{\gamma[B,X,f]}$ just to simplify the description —the formal definition of $\mathcal{A}_{\gamma[B,X,f]}$ will contain one separate statespace for each distinct $\mathcal{A}_{\gamma[B',X',f']}$.

As an example, suppose we have $\gamma[B, X, f]$ with γ is as in Figure 5 (left picture), but with $X = \{⑤\}$ and $B = E[X]$ (also depicted in Figure 8-a). And suppose we have a semipath $\mathcal{G}_w$ for a word $a_1 \cdots a_n \in (\mathbb{A}^{\pm})^*$. What does a mapping from an expansion of γ to $\mathcal{G}_w$ look like? For example, if the word is $abb^-aab^-a^-bb$ and the expansion is obtained by choosing words as in the left of Figure 6, we could obtain a mapping as shown on the right. Note that the

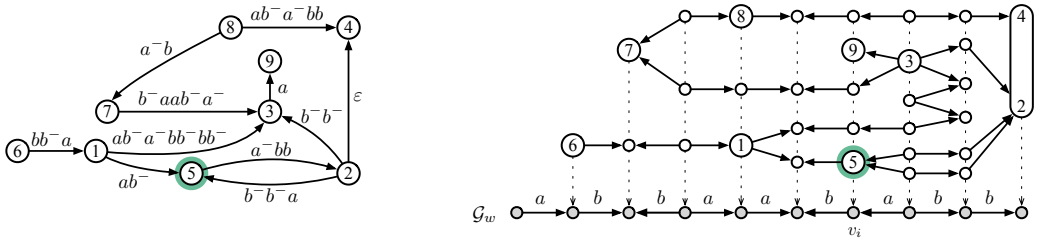


Figure 6 An expansion of $\gamma[B, X, f]$ and its homomorphic mapping into $\mathcal{G}_w$.

automaton has to somehow guess both the expansion and the mapping. Of course, already guessing the order of appearance of the homomorphic images of the vertices of γ (in our case, from left to right: ⑥, ⑦, {⑧, ①}, {⑨, ⑤}, ③, {④, ②}) would already yield an exponential automaton, regardless of bounded bridgewidth, and we therefore need to avoid this kind of brute-force guessing. Nevertheless, the automaton can first guess the non-output vertices that will appear to the left and to the right, as in Figure 7, and then, based on this guessing, find a way of decomposing the query into other, simpler, queries (as shown to the right). While there are exponentially many ways of guessing, the statespace remains polynomial if the bridgewidth is bounded. In the concrete case of the query of Figure 7, the automaton

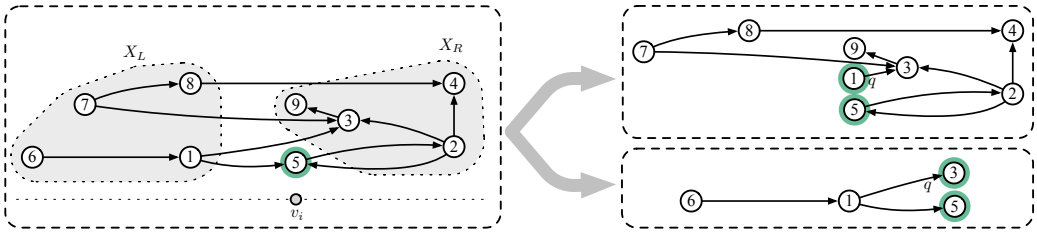


Figure 7 The automaton guesses the vertices mapped to the left (X_L) and to the right (X_R) of the current position v_i as well as a state q, and it decomposes the query into a conjunction of the two smaller queries on the right.

decomposes it by guessing a state $q \in Q$ (intuitively, the state of the automaton for the edge ① → ③ at the position where ⑤ is mapped) and running the conjunction of the two smaller

queries on the right. There may be many ways of decomposing them, but not too many, since the simpler queries will still be of the form $\gamma[B, X, f]$.

Suppose that $\mathcal{A}_{\gamma[B,X,f]}$ guesses a set of vertices X_R that will be mapped to the right of v_i, and a set of vertices X_L that will be mapped to the left[3] of v_i (*i.e.*, the gray blobs of Figure 7). For any such given partition X_L, X, X_R of the vertices of $\gamma[B, X, f]$, $\mathcal{A}_{\gamma[B,X,f]}$ proceeds in different ways according to whether some of these sets are empty. In particular, if both X_L and X_R are non-empty, it will need to guess what is the state of some edges that 'fly' across the current position, that is, edges with one endpoint in X_L and one endpoint in X_R (in Figure 7, the edge from ① to ③). This is indeed an exponential operation (and hence the construction may take exponential time) but it will not be reflected in the statespace of the automaton if $bw(\gamma)$ is bounded, nor in the space needed to perform this operation.

Now more concretely. Once X_L, X_R are fixed, applying the decomposition of Lemma 7 on the bridge B of M with $\tilde{X} = V \setminus (X_L \cup X_R)$, X_L and X_R, we obtain a set of bridges $\{B_i\}_i$ and a partition $\{X_i\}_i$ of $X_L \cup X_R$. We divide these bridges into left and right bridges: $\mathcal{B}_L = \{B_i : B_i \cap B[X_L] \neq \emptyset\}$ and $\mathcal{B}_R = \{B_i : B_i \cap B[X_R] \neq \emptyset\}$. We guess any function $g : \bigcup(\mathcal{B}_L \cup \mathcal{B}_R) \to Q$ so that $g(e) = f(e)$ for every $e \in B$. For each left-bridge $\hat{B} \in \mathcal{B}_L$ we consider $\gamma[\hat{B}, X^L_{\hat{B}}, g_{\hat{B}}]$ where $X^L_{\hat{B}}$ is the $(X \cup X_R)$-side of $\hat{B}$, and $g_{\hat{B}}$ is g restricted to $\hat{B}$. Similarly, for each right-bridge $\hat{B} \in \mathcal{B}_R$ we consider $\gamma[\hat{B}, X^R_{\hat{B}}, g_{\hat{B}}]$ where $X^R_{\hat{B}}$ is the $(X \cup X_L)$-side of $\hat{B}$, and $g_{\hat{B}}$ is g restricted to $\hat{B}$. Let Z be the $(V \setminus X)$-side of B (*i.e.*, the side which is not X). Note that if $X_L \cap Z = \emptyset$, then $\mathcal{B}_L = \emptyset$ and $\mathcal{B}_R = \{B\}$; and similarly for $X_R \cap Z = \emptyset$.

Thus, if we guess X_L, X_R as in our example of Figure 7, and if g is guessed so that for the edge e_{13} from ① to ③ we have $g(e_{13}) = q$, we generate the two queries on the right picture. Note that there is an expansion and homomorphism in accordance with the guessing X_L, X_R, g and sending ⑤ to v_i if and only if there are homomorphisms from some expansions of the two smaller queries on the right, mapping ① and ⑤ [resp. ③ and ⑤] to v_i. This describes the idea of the most interesting case: how to cover homomorphisms that map some vertices of Z to the left of v_i and some to the right of v_i (*i.e.*, $\mathcal{B}_L \neq \emptyset$ and $\mathcal{B}_R \neq \emptyset$). There are, however, two other remaining cases that need to be treated as well: (1) homomorphisms that map every vertex of Z to the right of v_i (*i.e.*, $Z \subseteq X_R$), which corresponds to simply moving to the 'next position' $2(i+1)$ of w (representing v_{i+1} in $\mathcal{G}_w$) updating the function f accordingly; (2) symmetrically, when all vertices of Z are mapped to the left of v_i; and (3) homomorphisms where at least one vertex of Z is mapped to the current vertex v_i, in which case the query must also be updated. We now describe all these cases formally.

1 $|\mathcal{B}_L \cup \mathcal{B}_R| > 1$. This is the case in our example. In this case, using alternation the automaton verifies that $2i \in (\bigcap_{B \in \mathcal{B}_L} \mathcal{A}_{\gamma[B, X^L_B, g_B]} \cap \bigcap_{B \in \mathcal{B}_R} \mathcal{A}_{\gamma[B, X^R_B, g_B]})(w)$. For example, this is the first kind of transition in our example of Figure 8, both in the transitions $a \to b$ and $c \to d$.

2 $|\mathcal{B}_L \cup \mathcal{B}_R| = 1$. In this case we have necessarily that either $X_L \cap Z$ or $X_R \cap Z$ is empty. Further, $\mathcal{B}_L \cup \mathcal{B}_R = \{B\}$. The automaton proceeds as follows: first it reads the loop information updating the states (case **A** below) and then it either moves right or left to the $2(i+1)$ or $2(i-1)$ position updating the states in f (cases **B1**, **B2**), or it guesses that there is a vertex that is mapped to vertex v_i of $\mathcal{G}_w$ and produces a new decomposition into bridges generating the alternation of smaller queries (**B3**). Here are the details:

[3] The vertices not in X that are mapped *at the same position* as those in X could be placed either in X_L or X_R (*e.g.*, in the picture ⑨ is in X_R).

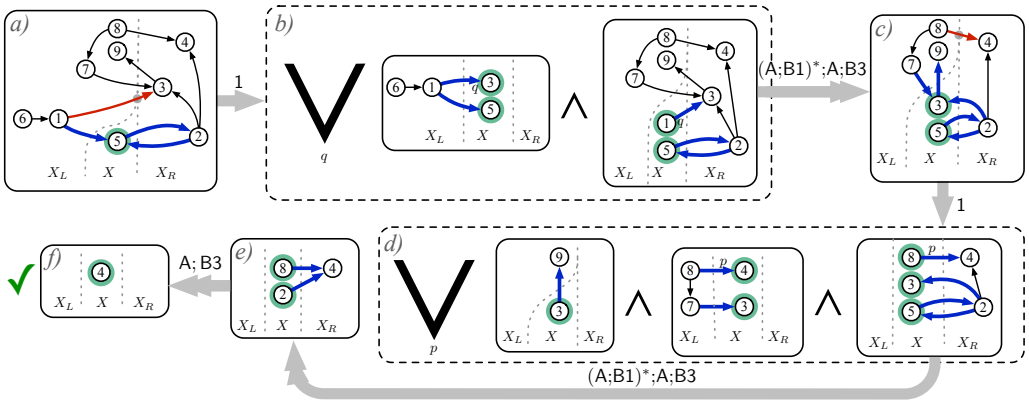


Figure 8 Schematic idea of a branch of an accepting run of $\mathcal{A}_{\gamma[B,X,f]}$ as witnessed by the homomorphism of Figure 6, and how the decomposition is done in terms of the $\gamma[B', X', f']$'s. Arrows are labelled by the type of transitions according to the cases **1**, **2**, **A**, **B1**, **B2**, **B3**; double-arrows stand for a succession of (non-alternating) transitions being applied.

A The automaton first reads L_i to the right (*i.e.*, the letter at position $2i$ in w) updating, nondeterministically, the states of f according to the loop type; that is, we replace f with f' for any $f' : B \to Q$ satisfying $(f(e), f'(e)) \in L_i$ for every $e \in B$. Then, it goes back left to the original position $2i$.

B Then it performs one of the three following actions, non-deterministically chosen:

B1 It checks $Z \cap X_R \neq \emptyset$ and moves right to position $2(i+1)$. That is, it reads L_i and then a_{i+1} to the right to end up at position $2(i+1)$ of w. It now updates the states of f' according to the label a_{i+1}; that is, it guesses some f'' where for every $e \in B$ we have $f''(e) = q$ for some $(+1, a_{i+1}, q)$ in $\vec{\delta}(f'(e))$. It finally verifies $2(i+1) \in \mathcal{A}_{\gamma[B,X,f'']}(w)$. For example, this is one of the actions implicit in the arrow $b \to c$ of Figure 8.

B2 It checks $Z \cap X_L \neq \emptyset$ and moves left to position $2(i-1)$. That is, it reads a_i and then L_{i-1} to the left to end up in position $2(i-1)$ of w. It now updates the states of f' accordingly; that is, it guesses some f'' where for every $e \in B$ we have $f''(e) = q$ for some $(-1, a_i, q)$ in $\overleftarrow{\delta}(f'(e))$. It finally verifies $2(i-1) \in \mathcal{A}_{\gamma[B,X,f'']}(w)$.

B3 It guesses that there is a vertex v in the $(X_L \cup X_R)$-side of B that will be mapped to position i. That is, it non-deterministically chooses some $v \in X_L \cup X_R$ and verifies $I(e) \cap F(e) \neq \emptyset$ for every edge $e \in B[v]$. Consider now the separator $S = (B \cup E[v]) \setminus B[v]$. The automaton chooses, non-deterministically, a state assignment for all new edges $S \setminus B$. That is, it picks some assignment $g : S \to Q$ so that: $g(e) = f(e)$ for every $e \in B$; $g(e) \in I(e)$ if $\eta_1(e) = v$; and $g(e) \in F(e)$ if $\eta_2(e) = v$. Now, using Lemma 5, S can be decomposed into a disjoint union of bridges $S = \bigcup_{i \in I} B_i$ so that each bridge will cover an independent part of the remaining graph of $\gamma[B, X, f']$. Thus, using alternation, the automaton finally verifies $2i \in (\bigcap_{i \in I} \mathcal{A}_{\gamma[B_i, X_{B_i}, g_{B_i}]})(w)$, where X_{B_i} is the $(X \cup \{v\})$-side of B_i, and g_{B_i} is g restricted to B_i (in Figure 8, this case corresponds to transitions $b \to c$ and $d \to e$). In particular, if $I = \emptyset$, the automaton simply accepts the word (in Figure 8, transition $e \to f$).

Figure 8 contains an example of how these cases interact in a run. Summing up, the automaton $\mathcal{A}_{\gamma[B,X,f]}$ can be implemented using 4 states $q_0^{B',X',f'}, q_{\mathbf{A1}}^{B',X',f'}, q_{\mathbf{A2}}^{B',X',f'}, q_{\mathbf{B}}^{B',X',f'}$ for each possible $\gamma[B', X', f']$, plus one global final state $F = \{q_f\}$. The initial set of states is $I = \{q_0^{B,X,f}\}$. For every possible $\mathcal{B}_L$, $\mathcal{B}_R$, if condition **1** holds, it uses alternation on states

$q_0^{B',X',f'}$ for suitable B', X', f'. Otherwise, it uses: states $q_{\mathbf{A1}}^{B,X,f'}, q_{\mathbf{A2}}^{B,X,f'}$ to perform the transitions described in **A** (*i.e.*, move to $2i+1$ with state $q_{\mathbf{A1}}^{B,X,f'}$ for some f', and then back to $2i$ with state $q_{\mathbf{A2}}^{B,X,f'}$); states $q_{\mathbf{B}}^{B,X,f'}, q_0^{B,X,f''}$ to move finally to position $2(i+1)$ or $2(i-1)$ as in **B1** or **B2** with the updated f''; and states $q_0^{B',X',f'}$'s to perform **B3** if possible. Whenever it accepts, it shifts to state q_f, from which it moves to the rightmost position to accept the word.

5.4 Correctness

For any subquery $\gamma[B, X, f]$ we prove, by induction on the size of $\gamma[B, X, f]$, that the automaton $\mathcal{A}_{\gamma[B,X,f]}$ satisfies (†). Suppose B separates $\tilde{X}$ and $V \setminus \tilde{X}$ in M, for $X \subseteq \tilde{X}$.

The *base case* is when $|V \setminus \tilde{X}| = 1$. Note that in this case, by construction, $\mathcal{A}_{\gamma[B,X,f]}$ works as a non-deterministic two-way automaton (that is, there is no alternation).
⇒) Suppose $2i \in \mathcal{A}_{\gamma[B,X,f]}(w)$. An accepting run consists basically on a number of applications of: the pair of back-and-forth transitions described in **A**, plus a transition from **B1** or **B2**, until by **B3** the automaton accepts. Assuming the word w is correctly labeled (*i.e.*, $w \in \mathcal{A}_{loop}$), such a run induces a homomorphism from an expansion of $\gamma[B, X, f]$ (given by the letters and loops read along) into $\mathcal{G}_w$ that maps every $x \in X$ to v_i. Notice that the last step in the accepting run is a collapse of the vertex of $V \setminus \tilde{X}$ by condition **B3** where $I = \emptyset$.
⇐) On the other hand, if there is an expansion of $\gamma[B, X, f]$ with a homomorphism mapping every $x \in X$ to v_i, then the sole vertex of $V \setminus \tilde{X}$ is mapped either to the right or to the left of i. In the former case the automaton can perform a number of **A**,**B2** steps until reaching the desired position and finishing with an application of **A**,**B3** steps accepting, and in the latter case it performs **A**,**B1** a number of times and then accepting with **A**,**B3**. Observe that all the loops can be "factored away" by using the information in the L_i's. Hence, if there is such a homomorphism, the automaton has an accepting run from position $2i$ on w.

The *inductive case* is when $|V \setminus \tilde{X}| > 1$.
⇒) Suppose $2i \in \mathcal{A}_{\gamma[B,X,f]}(w)$. If the first transition of the accepting run comes from **1**, then by the inductive hypothesis we obtain $(v_i, \ldots, v_i) \in \gamma[B, X, f](\mathcal{G}_w)$ using the following claim.

▷ Claim. If there are expansions of $\gamma[B, X_B^L, g_B]$ and $\gamma[B, X_B^R, g_B]$ with homomorphisms to $\mathcal{G}_w$ mapping every vertex of $\bigcup_{B \in \mathcal{B}_L \cup \mathcal{B}_R} X_B^L \cup X_B^R$ to v_i, then there is an expansion of $\gamma[B, X, g]$ mapping every vertex of X to v_i. In fact, the desired homomorphism and expansion is simply obtained as the union of the homomorphisms and expansions. This is because Lemma 7 partitions the graph of $\gamma[B, X, f]$ using pairwise disjoint sets of bridges $\mathcal{B}_L$ and $\mathcal{B}_R$.

Otherwise, the accepting run of the automaton consists of a number of applications of steps **A**,**B1** or **A**,**B2**, followed by the application of steps **A**,**B3** (this last one using alternation). Again, since the alternation is performed on disjoint sets of bridges and graph connected components by Lemma 5, once we have expansions and homomorphisms for these by inductive hypothesis, we can build an expansion of $\gamma[B, X, f]$ using the information on the applications of **A**,**B1** or **A**,**B2** that preceded this last **B3** step.
⇐) If there is an expansion of $\gamma[B, X, f]$ with a homomorphism h mapping every $x \in X$ to v_i, then the accepting run will depend on how the vertices are mapped to $\mathcal{G}_w$. If the $(V \setminus X)$-side of B has vertices that map through h both to the left and to the right of v_i in $\mathcal{G}_w$, then we follow **1** and we decompose into the conjunction of some $\gamma[B', X', f']$'s as directed by Lemma 7. Again, since the existence of h implies the existence of homomorphisms from expansions of the simpler $\gamma[B', X', f']$'s, it follows, by inductive hypothesis, that there are accepting runs for each $\mathcal{A}_{\gamma[B',X',f']}$ starting in $2i$, which in turns means that there is

an accepting run of $\mathcal{A}_{\gamma[B,X,f]}$ starting in $2i$. If, on the other hand, all vertices from the $(V \setminus X)$-side of B are mapped through h to the right of v_i (or all to the left) in $\mathcal{G}_w$, we follow the strategy to either go right (or left) by repeated applications of the transitions **A**,**B1** (or **A**,**B2**) until we arrive at some position $2(i+\ell)$ (or $2(i-\ell)$) to which some other vertex is mapped, and by **A**,**B3** we use alternation on some $\gamma[B', X', f']$ as directed by Lemma 5. Similarly as before, the homomorphism h and expansion implies the existence of homomorphisms from expansions of the $\gamma[B', X', f']$'s from $2(i+\ell)$, which in turns means that there is an accepting run from $2i$ for $\mathcal{A}_{\gamma[B,X,f]}$.

6 Lower bounds

The lower bounds of Theorem 1 are straightforward from known results. We give the proof ideas here.

The PSpace-hardness of point 1 of Theorem 1 is a consequence of a straightforward reduction from the containment problem for regular languages. Given two regular languages L_1, L_2 over an alphabet $\mathbb{A}$, given any two distinct symbols $\#, \perp$ not in $\mathbb{A}$, and given a graph $M \in \mathscr{C}$ with some edge e between two vertices u_1, u_2, consider the Boolean queries $\gamma_1, \gamma_2 \in \mathrm{CRPQ}(\mathscr{C})$ whose underlying graph is M over the alphabet $\mathbb{A} \dot\cup \{\perp, \#\}$, where γ_i has the language $\# \cdot L_i \cdot \#$ at edge e and the language $\{\perp\}$ at every other edge. It follows that γ_1 is contained in γ_2 if and only if $L_1 \subseteq L_2$.

The ExpSpace-hardness of point 2 follows from a reduction from the following containment problem restricted to two-vertex Boolean CRPQ, which is ExpSpace-hard.

▶ **Lemma 8.** *The problem of deciding, given a Boolean CRPQ γ_1 with two vertices and one edge and another Boolean CRPQ γ_2 with two vertices and arbitrarily many edges in the same direction, whether γ_1 is contained in γ_2, is* ExpSpace*-hard.*

We show this lemma by an easy adaptation of the proofs of [7, Theorem 6] and [5, Lemma 14].

Proof. We reduce from the following 2^n-tiling problem, which is ExpSpace-complete (see, *e.g.*, [10, Theorem 6.1]). An input instance consists of a number $n \in \mathbb{N}$ written in unary, a finite set Δ of tiles, two relations $H, V \subseteq \Delta \times \Delta$ specifying constraints on how tiles should be placed horizontally and vertically, and the starting and final tiles $t_S, t_F \in \Delta$. A solution to the input instance is a 'consistent' assignment of tiles to a finite rectangle having 2^n columns. Concretely, a solution is a function $f : \{1, \ldots, 2^n\} \times \{1, \ldots, k\} \to \Delta$, for some $k \in \mathbb{N}$, such that $f(1,1) = t_S$, $f(2^n, k) = t_F$, and $f((i,j), f(i+1,j)) \in H$ and $f((i,j), f(i,j+1)) \in V$ for every i, j in range. We now show the following.

▷ Claim 9. For every 2^n-tiling problem T there are Boolean CRPQ γ_1, γ_2, computable in polynomial time from T, such that $\gamma_1 \subseteq \gamma_2$ if, and only if, T has no solution. Further, γ_1 is of the form $\exists x, y\ x \xrightarrow{L} y$ and γ_2 is of the form $\exists x, y\ \bigwedge_{0 \le i \le n} (x \xrightarrow{L_i} y)$, where each L_i is given as a regular expression.

For any tiling instance as above, we show how to define the two CRPQ over the alphabet $\mathbb{A} := \Delta \cup \{0, 1, \#\}$ so that containment fails if and only if there is a solution to T. We will encode a solution of a tiling as a word of $\#((0+1)^n \cdot \Delta)^*\#$, where the rectangle of tiles is read left-to-right and top-to-bottom, and each block (*i.e.*, each element of $(0+1)^n\ \Delta$) represents the column number (in binary) and the tile. The symbols $\#$ at the beginning and end of the word are used for technical reasons.

For enforcing this encoding, we define regular languages E, F_C, F_H and G_i for each $i \le n$ over $\mathbb{A}$.

The language E gives the general shape of the encoding of solutions,

$$E = \#\, 0^n\ t_S\ ((0+1)^n \Delta)^*\ 1^n\ t_F\, \#,$$

in particular it enforces that it starts and ends with the correct tiles. The language F_C detects adjacent blocks with an error in the column number bit, which can be easily defined with a polynomial size NFA (*e.g.*, if $n = 3$ in particular F_C contains every word having '$001\, t\, 011$', '0000' or '$t\, t'$' as factor, for $t, t' \in \Delta$). The language F_H checks that there are adjacent blocks in which the tiles do not respect the horizontal adjacency relation H,

$$F_H = \bigcup_{(t_1,t_2)\in\Delta^2\setminus H} t_1\ \overline{0^n}\ t_2,$$

where $\overline{0^n} = (0+1)^n \setminus \{0^n\}$. Finally, $G_0, \ldots, G_n$ are used to check that there are two blocks at distance 2^n which do not respect the vertical adjacency relation V; in other words, there is a factor of the word whose first and last blocks have the same column number, it contains not more than one block with column number 1^n (otherwise we would be skipping a row), and its first and last tiles are not V-related. First, G_0 checks that the first and last blocks of the factor we are interested in do not conform to V, and furthermore that there is exactly one column number 1^n in between

$$G_0 = \bigcup_{(t_1,t_2)\in\Delta^2\setminus V} (0+1)^n\, t_1\, (\overline{1^n}\, \Delta)^*\, 1^n\, (\Delta\, \overline{1^n})^*\, t_2,$$

where $\overline{1^n} = (0+1)^n \setminus \{1^n\}$. For each $b \in \{0,1\}$ and $i \in \{1, \ldots, n\}$ we define G_i^b to check that the i-th bit of the address of both the first and last tile is set to b,

$$G_i^b = (0+1)^{i-1}\, b\, (0+1)^{n-i}\, \Delta\, \big((0+1)^n\, \Delta\big)^*\, (0+1)^{i-1}\, b\, (0+1)^{n-i}\, \Delta,$$

and we define G_i as $G_i^0 + G_i^1$. For each one of these languages one can produce a regular expression recognizing the language in polynomial time. Finally, the Boolean CRPQ are

$$\gamma_1 = \exists x, y\ \ x \xrightarrow{E} y \qquad\qquad \gamma_2 = \exists x, y\ \bigwedge_{0\le i\le n} x \xrightarrow{G_i \cup F_C \cup F_H} y.$$

Observe that $\gamma_1 \subseteq \gamma_2$ iff every word of E contains an error in the encoding (*i.e.*, it contains a word from F_C as factor), or it contains a pair of horizontally adjacent tiles which do not respect the horizontal constraints (*i.e.*, it contains a word from F_H as factor) or, otherwise, it contains a pair of vertically adjacent tiles which do not respect the vertical constraints (*i.e.*, it contains a word from $\bigcap_{0\le i\le n} G_i$ as factor). Thus, $\gamma_1 \subseteq \gamma_2$ if, and only if, T has no solution. ◀

Now, in view of Lemma 8, suppose γ_1 reads language $L \subseteq \mathbb{A}^*$ at its only edge, and γ_2 has ℓ edges with languages $L_1, \ldots, L_\ell \subseteq \mathbb{A}^*$. By the characterization of Lemma 3, it follows that for every k, $\mathscr{C}$ contains a graph M with a minor from $\mathcal{E}_k$. Hence, there is some graph $M \in \mathscr{C}$ and two sets of vertices V_1, V_2 therein so that $M|_{V_1}$ and $M|_{V_2}$ are connected and M has ℓ distinct edges $e_1, \ldots, e_\ell$ with source in V_1 and target in V_2. Consider the following two Boolean CRPQ($\mathscr{C}$) γ_1', γ_2' having M as underlying graph over the alphabet $\mathbb{A} \dot\cup \{\bot\}$, where $\bot$ is some symbol not in $\mathbb{A}$. First, γ_1' is defined as follows: every edge in $M|_{V_1}$ or $M|_{V_2}$ is labelled with $\{\varepsilon\}$; every edge e_i is labelled with L; and every other edge is labelled with $\{\bot\}$. Second, γ_2' is defined as follows: every edge in $M|_{V_1}$ or $M|_{V_2}$ is labelled with $\{\varepsilon\}$; every edge e_i is labelled with L_i; and every other edge is labelled with $(\mathbb{A} \cup \{\bot\})^*$. It is not hard to see that $\gamma_1' \subseteq \gamma_2'$ if and only if $\gamma_1 \subseteq \gamma_2$. See Figure 9 for an example. It then follows that containment of Boolean CRPQ($\mathscr{C}$) is ExpSpace-hard.

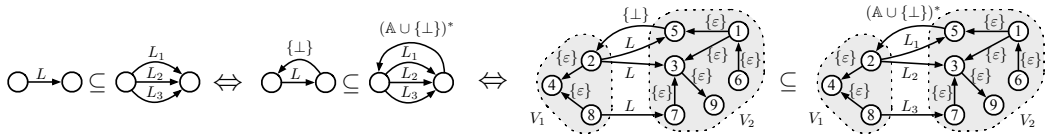


Figure 9 Reduction from two vertices Boolean CRPQ containment (left) to Boolean CRPQ($\mathscr{C}$) containment (right).

7 Final remarks

Observe that the following graph measures are all lower bounds for bridgewidth: maximum vertex degree, maximum number of pairwise edge-independent paths between two vertices (and hence also the size of a minimum cut set by Menger's theorem), and the graph's *treewidth*.[4] In particular, bounded bridgewidth implies bounded treewidth, and thus classes of UC2RPQ of bounded bridgewidth can be evaluated in polynomial time. Classes of UC2RPQ of bounded bridgewidth are somewhat more robust than those of bounded treewidth, in the sense that both the evaluation and containment problems are 'efficient'. This is what happens for classes of Conjunctive Queries of bounded treewidth, where both these problems —which in this case are essentially the same— are polynomial-time computable [13, 12]. On the other hand, bounded treewidth does not imply bounded bridgewidth (take for instance $\bigcup_i \mathcal{E}_i$). As for treewidth [3], the problem of whether a graph has bridgewidth k (where both the graph and k are input) is NP-complete, by a simple reduction from the MAX-CUT problem (see, *e.g.*, [1]).

What about boundedness? We conjecture that an adaptation of the bridgewidth measure may also yield a similar result for the boundedness problem for UCRPQ (note: no 2-wayness). More concretely, let the **scc-minor** of a graph M be the result of contracting all edges belonging to the same strongly connected component (scc). Note that the resulting graph is a directed acyclic graph (possibly with self-loops). The **scc-bridgewidth** of M is the bridgewidth of the scc-minor of M.

▷ Conjecture. For every class $\mathscr{C}$ of graphs,

1. if $\mathscr{C}$ has bounded scc-bridgewidth, the UCRPQ($\mathscr{C}$)-boundedness problem is in PSPACE;
2. if $\mathscr{C}$ has unbounded scc-bridgewidth, the UCRPQ($\mathscr{C}$)-boundedness problem is EXPSPACE-complete.

References

1 Max cut problem between two connected subgraphs. Stack Exchange. Version: 2018-07-26 21:13. URL: `https://cstheory.stackexchange.com/q/41267/49964`.

2 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, 50(5):68:1–68:40, 2017. `doi:10.1145/3104031`.

3 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

4 Pablo Barceló. Querying graph databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 175–188, 2013. `doi:10.1145/2463664.2465216`.

[4] For treewidth, the root of the tree decomposition is a bag containing a vertex whose set of incident edges is a bridge (it exists due to Lemma 6), together with all its neighbors. We then apply iteratively Lemma 4 to decompose the remaining graph into subtrees.

5 Pablo Barceló, Diego Figueira, and Miguel Romero. Boundedness of conjunctive regular path queries. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 104:1–104:15. Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.104.

6 Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. Semantic acyclicity on graph databases. *SIAM J. Comput.*, 45(4):1339–1376, 2016. doi:10.1137/15M1034714.

7 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Principles of Knowledge Representation and Reasoning (KR)*, pages 176–185, 2000.

8 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query answering and query containment over semistructured data. In *International Symposium on Database Programming Languages (DBPL)*, volume 2397 of *Lecture Notes in Computer Science*, pages 40–61. Springer, 2001. doi:10.1007/3-540-46093-4_3.

9 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. *Journal of Computer and System Sciences (JCSS)*, 64(3):443–465, 2002. doi:10.1006/jcss.2001.1805.

10 Bogdan S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences (JCSS)*, 32(3):374–392, 1986. doi:10.1016/0022-0000(86)90036-X.

11 Daniela Florescu, Alon Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 139–148. ACM Press, 1998. doi:10.1145/275487.275503.

12 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

13 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Symposium on Theory of Computing (STOC)*, pages 657–666. ACM Press, 2001. doi:10.1145/380752.380867.

14 Juan L. Reutter, Miguel Romero, and Moshe Y. Vardi. Regular queries on graph databases. *Theoretical Computer Science*, 61(1):31–83, 2017. doi:10.1007/s00224-016-9676-2.