Expressive Languages for Path Queries over Graph-Structured Data

Pablo Barceló
Dept. of Computer Science, Univ. of Chile
pbarcelo@dcc.uchile.cl

Carlos Hurtado
Fac. Ingeniería y Ciencias, Univ. A. Ibañez
carlos.hurtado@uai.cl

Leonid Libkin
Sch. of Informatics, Univ. of Edinburgh
libkin@inf.ed.ac.uk

Peter Wood
Dept. of CS and Inf. Syst., Birkbeck, U. London
ptw@dcs.bbk.ac.uk

ABSTRACT

For many problems arising in the setting of graph querying (such as finding semantic associations in RDF graphs, exact and approximate pattern matching, sequence alignment, etc.), the power of standard languages such as the widely studied conjunctive regular path queries (CRPQs) is insufficient in at least two ways. First, they cannot output paths and second, more crucially, they cannot express *relations* among paths.

We thus propose a class of *extended* CRPQs, called ECRPQs, which add regular relations on tuples of paths, and allow path variables in the heads of queries. We provide several examples of their usefulness in querying graph structured data, and study their properties. We analyze query evaluation and representation of tuples of paths in the output by means of automata. We present a detailed analysis of data and combined complexity of queries, and consider restrictions that lower the complexity of ECRPQs to that of relational conjunctive queries. We study the containment problem, and look at further extensions with first-order features, and with non-regular relations that express arithmetic properties of paths, based on the lengths and numbers of occurrences of labels.

Categories and Subject Descriptors. H.2.1 [**Database Management**]: Logical Design—*Data Models*; F.1.1 [**Computation by abstract devices**]: Models of Computation—*Automata*

General Terms. Theory, Languages, Algorithms

Keywords. Graph databases, conjunctive queries, regular relations, regular path queries

PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.

1. Introduction

For graph-structured data, queries that allow users to specify the types of paths in which they are interested have always played a central role. Most commonly, the specification of such paths has been by means of regular expressions over the alphabet of edge labels [2, 10, 13, 16, 29]. The output of a query is typically a set of tuples of nodes that are connected in some way by the paths specified. The canonical class of queries with this functionality are the conjunctive regular path queries (CRPQs), which have been the subject of much investigation, e.g. [10, 14, 16].

However, the rapid increase in the size and complexity of graph-structured data (e.g. in the Semantic Web, or in biological applications) has raised the need for additional functionality in query languages. Specifically, in many examples, the minimum requirements of sufficiently expressive queries are: (a) the ability to define complex semantic relationships between paths and (b) the ability to include paths in the output of the query. Neither of these is supported by CRPQs.

There are multiple examples of queries that require these new capabilities. For example, [5] introduces a query language for RDF/S in which paths can be compared based on specific semantic associations. In handling biological sequences one often needs to compare paths based on similarity (e.g., edit distance) [20]. Paths can be compared with respect to other parameters, e.g., lengths or numbers of occurrences of labels, which can be useful in route-finding applications [6].

As for the ability to output paths, this has been proposed, for example, as an extension to the SPARQL query language – the standard for retrieving RDF data [24]. However, [24] only proposed a declarative language, and left most basic questions unexplored (e.g., what should an output be if there are infinitely many paths between nodes?). Other applications for this new functionality include determining the provenance of data or artifacts [21], finding associations in linked data [27], biological data [26] or social (or criminal) net-

works [32], as well as performing semantic searches over web-derived knowledge [36].

While the need for the extended functionality of graph query languages is well-documented (and sometimes is even incorporated into a programming syntax), the basic theoretical properties of such languages are completely unexplored. We do not know whether queries can be meaningfully evaluated, what their complexity is, whether they can be optimized, etc.

Our main goals, therefore, are to formally define extensions of graph queries that can express complex semantic associations between paths and output paths to the user, and to study them, concentrating on query evaluation and its complexity, as well as some static analysis problems.

We work with the class of *extended conjunctive regular path queries* or (ECRPQs), which generalize CRPQs by allowing them to express the kind of semantic association properties we explained above. That is, we allow (i) n-tuples of path labels to be checked for conformity to n-ary path languages, and (ii) paths, rather than simply nodes, to be output. Conformity with respect to n-ary languages is given, following the idea behind CRPQs, with respect to n-ary regular relations.

As an example, consider a graph G with a single edge label, defining the student-advisor relationship. Using CRPQs, one can express many queries, such as finding academic ancestors, or people whose sets of academic parents and grandparents intersect, or checking whether Van Gucht and Tannen have a common academic ancestor (and if so, who that person is). However, with CRPQs we *cannot* express queries asking for pairs of scientists who have the same-length path to Tarski, for example, nor can one ask for the precise paths by which Van Gucht and Tannen are related to their common academic ancestor. With ECRPQs, we *can* express such queries.

While leaving the above queries to the reader as an exercise, we now outline a few examples of problems where the power of ECRPQs is required. They will be fully developed in Section 3, after we have presented the syntax and semantics of ECRPQs.

(i) Pattern matching Given an alphabet Σ and a set of variables $\mathcal{V}$, a *pattern* is a string over $\Sigma \cup \mathcal{V}$. A pattern defines a pattern language by instantiating variables with strings in Σ^*. Pattern languages need not be context-free: e.g., the language of squared words over Σ can be expressed by the pattern XX, where $X \in \mathcal{V}$. But finding nodes x and y connected by a path whose label is in the language of squared words can be expressed by the ECRPQ:

$$Ans(x, y) \leftarrow (x, \pi_1, z), (z, \pi_2, y), \pi_1 = \pi_2$$

where x, y and z are node variables and π_1 and π_2 are path variables. Variables z, π_1, and π_2 are meant to be existentially quantified. What makes this different from CRPQs is the binary relation $\pi_1 = \pi_2$ on paths: it states that the paths between x and an intermediate node z, and between z and y are the same.

(ii) Semantic web associations In RDF/S, properties can be declared to be subproperties of other properties. This is used in [5] to define a notion of semantic association based on *ρ-isomorphic* property sequences: two sequences are ρ-isomorphic if they are of the same length and the properties at the same position in each sequence are subproperties of one another. Such pairs of sequences can be found by a modification of the previous query with a different binary relation expressing the fact that the paths are ρ-isomorphic.

(iii) Approximate matching Approximate string matching [19, 23] and (biological) sequence alignment [20] are both based on the notion of edit distance. The relation representing pairs of sequences that have edit distance at most k from one another, for some fixed k, is regular [18]. So given a graph representing a pair of sequences, an ECRPQ can determine whether they have edit distance at most k. We show in Section 3.1 that we can also output the actual gaps and mismatches in the sequences using an ECRPQ.

Outline of the results After we formally define ECRPQs, we present an algorithm for query evaluation. It turns out that the sets of labels of paths satisfying a query are regular, and thus the evaluation algorithm constructs automata to represent such sets.

We then investigate the complexity of query evaluation. As yardsticks, we consider relational languages as well as CRPQs. For conjunctive queries, combined complexity is NP-complete, while it jumps to Pspace-complete for relational calculus. Hence we cannot hope to get anything below NP for ECRPQs, and we hope not to exceed the complexity of relational queries in a reasonable class. As for data complexity, it is known to be NLogspace-complete for CRPQs, so this will serve as another benchmark.

It turns out that the data complexity of ECRPQs matches that of CRPQs, but combined complexity goes up from NP to Pspace, matching relational calculus instead. In this case it is natural to look for restrictions. A standard one for CQs is a restriction to acyclicity. This works for CRPQs – combined complexity becomes tractable – but does not work for ECRPQs, as the combined complexity remains Pspace-complete. However, if our regular relations can only talk about lengths of paths, then the complexity of ECRPQs drops to NP, matching the complexity of the usual relational CQs.

We then look at extensions of CRPQs and ECRPQs: with negation and universal quantification, and with some non-regular relations. For the former, we get surprisingly reasonable bounds for CRPQs, but the complexity becomes too high when both negation and relations on paths are allowed. For the latter, we look at extensions with linear constraints on path lengths,

and prove some good complexity bounds (tractable data complexity and NP combined complexity). We also look at relations that compare numbers of occurrences of labels in paths, and prove some low complexity bounds for queries with such relations.

While query containment is known to be decidable for CRPQs, we show that ECRPQs share more properties with full relational calculus: containment for them becomes undecidable. We recover decidability in one important subcase though.

Organization In the next section, we present background material on graphs, regular relations and CRPQs. Section 3 introduces ECRPQs and looks at their applications in more detail. In Section 4, we consider the evaluation of ECRPQs. Section 5 deals with the data and combined complexity of ECRPQs. In Section 6 we look at query containment, and in Section 7 we consider extensions with negation, and with non-regular features.

2. Preliminaries

Labeled graphs and paths Queries in our setting will be evaluated over labeled *database* graphs (*db*-graphs), that naturally model semistructured data. Formally, if Σ is a finite alphabet, then a Σ-*labeled db-graph* G (or simply *db*-graph if Σ is clear from the context) is a pair (V, E), such that V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of directed edges labeled in Σ.

A *path* ρ between nodes v_0 and v_m in G is a sequence $v_0 a_0 v_1 a_1 v_2 \cdots v_{m-1} a_{m-1} v_m$, where $m \geq 0$, so that all the v_i's are in V, all the a_j's are letters of Σ, and (v_i, a_i, v_{i+1}) is in E for each $i < m$. The *label* of such a path ρ, denoted by $\lambda(\rho)$, is the string $a_0 \cdots a_{m-1} \in \Sigma^*$. We also define the empty path as (v, ε, v) for each $v \in V$; the label of such a path is the empty string ε.

Note that a Σ-labeled *db*-graph G can be naturally viewed as a nondeterministic finite automaton (NFA) over alphabet Σ without initial and final states. Its states are nodes in V, and its transitions are edges in E. We use this equivalence in several constructions in the paper.

Regular relations As our plan is to extend the notion of recognizability from string languages to n-ary string relations, we now give the standard definition of *regular* relations over Σ [15, 18, 8]. Let $\perp$ be a symbol not in Σ. We denote the extended alphabet $(\Sigma \cup \{\perp\})$ by $\Sigma_\perp$. Let $\bar{s} = (s_1, \ldots, s_n)$ be an n-tuple of strings over alphabet Σ. We construct a string $[\bar{s}]$ over alphabet $(\Sigma_\perp)^n$, whose length is the maximum of the s_j's, and whose i-th symbol is a tuple $(c_1, \ldots, c_n)$, where each c_k is the i-th symbol of s_k, if the length of s_k is at least i, or $\perp$ otherwise. In other words, we pad shorter strings with the symbol $\perp$, and then view the n strings as one string over the alphabet of n-tuples of letters. An n-ary relation S on Σ^* is *regular*, if the set $\{[\bar{s}] \mid \bar{s} \in S\}$ of strings over alphabet $(\Sigma_\perp)^n$ is regular (i.e., accepted by an automaton over $(\Sigma_\perp)^n$, or given by a regular expression over $(\Sigma_\perp)^n$). We shall often use the same letter for both a regular expression over $(\Sigma_\perp)^n$ and the relation over Σ^* it denotes, as doing so will not lead to any ambiguity.

As an example, consider a binary relation $s \preceq s'$, saying that s is a prefix of s'. The automaton recognizing this relation accepts if it reads a sequence of letters of the form (a, a), for $a \in \Sigma$, possibly followed by a sequence of letters of the form $(\perp, b)$, for $b \in \Sigma$. As another example, consider a binary relation $\mathrm{el}(s, s')$ (equal length) saying that $|s| = |s'|$. This relation is recognized by an automaton that accepts if it does not see any letters involving the $\perp$ symbol.

To understand which relations on strings are regular, it is often useful to provide a model-theoretic characterization of this class. In the following we assume familiarity with first-order logic (FO). Consider the FO-structure $\mathcal{M}_{\mathrm{univ}} = \langle \Sigma^*, \preceq, \mathrm{el}, (P_a)_{a \in \Sigma} \rangle$ with domain Σ^*, where $\preceq$ and el are as above, and $P_a(s)$ is true iff the last letter for s is a. This is known as a *universal automatic structure* due to the following [8, 9]: an n-ary relation S on Σ^* is regular iff there exists an FO formula $\phi_S(x_1, \ldots, x_n)$ over $\mathcal{M}_{\mathrm{univ}}$ such that $S = \{\bar{s} \in (\Sigma^*)^n \mid \mathcal{M}_{\mathrm{univ}} \models \phi_S(\bar{s})\}$.

In particular, regular relations are closed under all Boolean combinations, product, and projection. Furthermore, using the above result it is quite easy to show that an n-ary relation is regular, by exhibiting FO formulae defining them (see [8, 9, 7] for examples). For example, $|s| < |s'|$ is a regular relation definable by $\phi(x, y) = \exists y' \ (y' \preceq y \wedge y' \neq y \wedge \mathrm{el}(y', x))$. On the other hand, more elaborate techniques have to be used to prove that an n-ary relation on Σ is *not* regular. Examples of this kind include the binary relation $\preceq_{ss}$, that consists of all pairs (s_1, s_2) such that s_1 is a subsequence of s_2, and the ternary relation that contains all tuples (s_1, s_2, s_3) such that $s_1 s_2 = s_3$.

Conjunctive regular path queries A basic querying mechanism for graph databases is the class of *regular path queries* [3, 11] that retrieve all pairs of objects in a *db*-graph connected by a path conforming to some regular expression. However, it has been argued (e.g. [30]) that in order to make regular path queries useful in practice, they should be extended with several features, one of them being the possibility of using conjunctions of atoms. This extension yields the class of *conjunctive* regular path queries, which we formally define below (see also [13, 29, 16, 10]).

Fix a countable set of *node* variables (typically denoted by $x, y, z, \ldots$), and a countable set of *path* variables (denoted by $\pi, \omega, \chi, \ldots$). A *conjunctive regular path query* (CRPQ) Q over a finite alphabet Σ is an expression of

the form:

$$Ans(\bar{z}) \leftarrow \bigwedge_{1\leq i\leq m}(x_i, \pi_i, y_i), \bigwedge_{1\leq j\leq t} L_j(\omega_j), \qquad (1)$$

such that

(i) $m > 0$, $t \geq 0$,

(ii) each L_j is a regular expression over Σ,

(iii) $\bar{x} = (x_1, \ldots, x_m)$, $\bar{y} = (y_1, \ldots, y_m)$ and $\bar{z}$ are tuples of node variables,

(iv) $\{\pi_1, \ldots, \pi_m\}$ are distinct path variables,

(v) $\{\omega_1, \ldots, \omega_t\}$ are distinct path variables and each ω_j is among the π_i's, and

(vi) $\bar{z}$ is a tuple of node variables among $\bar{x}$ and $\bar{y}$.

The atom $Ans(\bar{z})$ is the *head* of the query, the expression on the right of the $\leftarrow$ is its *body*. The query Q is *Boolean* if its head is of the form $Ans()$, i.e. $\bar{z}$ is the empty tuple.

Intuitively, such a query Q selects tuples $\bar{z}$ for which there exist values of the remaining node variables from $\bar{x}$ and $\bar{y}$ and paths π_i between x_i and y_i whose labels satisfy the regular expressions L_1 to L_t. Formally, to define the semantics of CRPQs Q of the form (1), we first introduce a relation $(G, \sigma, \mu) \models Q$, where σ is a mapping from $\bar{x}, \bar{y}$ to the set of nodes of a *db*-graph $G = (V, E)$, and μ is a mapping from $\{\pi_1, \ldots, \pi_m\}$ to paths in G. This relation holds iff $\mu(\pi_i)$ is a path in G from $\sigma(x_i)$ to $\sigma(y_i)$, for $1 \leq i \leq m$, and the label of each path $\mu(\omega_j)$ is in the language of L_j, for $1 \leq j \leq t$.

We now define $Q(G)$ to be the set of tuples $\sigma(\bar{z})$ such that $(G, \sigma, \mu) \models Q$. If Q is Boolean, we let $Q(G)$ be true if $(G, \sigma, \mu) \models Q$ for some σ and μ (that is, as usual, the empty tuple models the Boolean constant `true`, and the empty set models the Boolean constant `false`).

Remark: Our syntax differs slightly from the usual CRPQ syntax in the literature (see e.g. [16, 10]). The reason is that we make explicit use of path variables in the queries – to treat CRPQs and ECRPQs in a uniform manner – while the standard approach is to refer to paths only implicitly.

3. Extended Conjunctive Regular Path Queries

Our goal is to extend the class of CRPQs in two ways. First, we want to allow *free path variables* in the heads of queries. Second, we want the bodies of queries to permit checking *relations on sets of paths* rather than just conformance of individual paths to regular languages. This leads to the definition of a class of *extended CRQPs*.

An *extended conjunctive regular path query* (ECRPQ) Q over Σ is an expression of the form:

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1\leq i\leq m}(x_i, \pi_i, y_i), \bigwedge_{1\leq j\leq t} R_j(\bar{\omega}_j), \qquad (2)$$

such that

(i) $m > 0$, $t \geq 0$,

(ii) each R_j is a regular expression that defines a regular relation over Σ,

(iii) $\bar{x} = (x_1, \ldots, x_m)$ and $\bar{y} = (y_1, \ldots, y_m)$ are tuples of node variables,

(iv) $\bar{\pi} = (\pi_1, \ldots, \pi_m)$ is a tuple of distinct path variables,

(v) $\{\bar{\omega}_1, \ldots, \bar{\omega}_t\}$ are distinct tuples of path variables, such that each $\bar{\omega}_j$ is a tuple of variables from $\bar{\pi}$, of the same arity as R_j,

(vi) $\bar{z}$ is a tuple of node variables among $\bar{x}, \bar{y}$, and

(vii) $\bar{\chi}$ is a tuple of path variables among those in $\bar{\pi}$.

Note that this is similar to the definition of CRPQs; the main differences between (1) and (2) are:

- ECRPQs can check whether a tuple of paths belongs to a regular relation, rather than just checking whether a path belongs to a regular language; and
- outputs of ECRPQs may contain both nodes and paths, while outputs of CRPQs contain only nodes.

The head, the body, and the notion of Boolean ECRPQs are defined in the standard way. The *relational* part of an ECRPQ Q (2) is $\bigwedge_{1\leq i\leq m}(x_i, \pi_i, y_i)$.

The semantics of ECRPQs is defined by a natural extension of the semantics of CRPQs. For an ECRPQ Q of the form (2), a *db*-graph G and mappings σ from node variables to nodes and μ from path variables to paths, we write $(G, \sigma, \mu) \models Q$ if

- $\mu(\pi_i)$ is a path in G from $\sigma(x_i)$ to $\sigma(y_i)$, for $1 \leq i \leq m$, and
- for each $\bar{\omega}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$, the tuple of strings consisting of labels of $\mu(\pi_{j_1}), \ldots, \mu(\pi_{j_k})$ belongs to the relation R_j.

The output of Q on G (where the head of Q is $Ans(\bar{z}, \bar{\chi})$) is defined as

$$Q(G) \;=\; \{(\sigma(\bar{z}), \mu(\bar{\chi})) \mid (G, \sigma, \mu) \models Q\}.$$

Note that the implicit existential quantification over path variables that appear in the body but not in the head is quantification over a potentially infinite set, as there are infinitely many paths in any cyclic *db*-graph.

From now on, we identify the class of CRPQs with the restriction of the class of ECRPQs to queries that do not

use regular relations of arity ≥ 2. This is more general than the definition of the previous section, since we now allow CRPQs to output paths.

It is easy to prove that the class of ECRPQs is strictly more expressive than the class of CRPQs. Formally,

Proposition 3.1. *There is a Boolean ECRPQ Q that is not equivalent to any CRPQ Q'.*

3.1 Applications of ECRPQs

In this section, we show that ECRPQs can express queries found in a wide variety of application areas, including finding associations in semantic web (or linked) data, pattern matching, approximate string matching, and biological sequence alignment.

Finding semantic web associations In a query language for RDF/S introduced in [5], paths can be compared based on specific *semantic associations*. Edges correspond to RDF properties and paths to property sequences. A property a can be declared to be a subproperty of property b, which we denote by $a \prec b$. Two property sequences u and v are called ρ-isomorphic iff $u = u_1, \ldots, u_n$ and $v = v_1, \ldots, v_n$, for some n, and $u_i \prec v_i$ or $v_i \prec u_i$ for every $i \leq n$ [5]. Nodes x and y are called ρ-isoAssociated iff x and y are the origins of two ρ-isomorphic property sequences.

Finding nodes which are ρ-isoAssociated cannot be done in a query language supporting only conventional regular expressions, not least because doing so requires checking that two paths are of equal length. However, pairs of ρ-isomorphic sequences can be expressed using the regular relation R given by the following regular expression: $\big(\bigcup_{a,b\in\Sigma:\ (a\prec b \vee b\prec a)}(a,b)\big)^*$. Then an ECRPQ returning pairs of nodes x and y that are ρ-isoAssociated can be written as follows:

$$Ans(x,y) \leftarrow (x,\pi_1,z_1),(y,\pi_2,z_2),R(\pi_1,\pi_2)$$

Path variables in an ECRPQ can also be used to return the actual paths found by the query, a mechanism found in the query languages proposed in [2, 5, 21, 24]. For example, in [5] a ρ-query can take a pair of nodes u, v and return the property sequences relating them. This too can be expressed by an ECRPQ:

$$Ans(\pi_1,\pi_2) \leftarrow (u,\pi_1,z_1),(v,\pi_2,z_2),R(\pi_1,\pi_2)$$

where the regular relation R is defined as above.

Pattern matching Let Σ be a finite alphabet and $\mathcal{V}$ be a countable set of variables such that $\Sigma \cap \mathcal{V} = \emptyset$. A *pattern* α is a string over $\Sigma \cup \mathcal{V}$. It denotes the language $L_\Sigma(\alpha)$ obtained by applying substitutions $\sigma : \mathcal{V} \to \Sigma^*$ to α. As we remarked already, such languages need not even be context-free.

However, for each pattern $\alpha = \alpha_1 \cdots \alpha_n$, where every $\alpha_i \in \Sigma \cup \mathcal{V}$, we can define an ECRPQ $Q_\alpha(x,y)$ which finds pairs of nodes connected by a path in $L_\Sigma(\alpha)$ (note that this property is not definable by a CRPQ).

Indeed, the relational part of Q_α is $(x_0,\pi_1,x_1),\ldots,(x_{n-1},\pi_n,x_n)$. If α_i is a letter, then Q_α contains the atom $a(\pi_i)$, and if α_i is a variable, then it contains $\Sigma^*(\pi_i)$. Finally, to ensure equality of variables, for every two α_i, α_j which are the same variable, the query Q_α contains a conjunct $\pi_i = \pi_j$. It is clear that Q_α indeed finds nodes connected by paths from $L_\Sigma(\alpha)$.

In fact, ECRPQs can express queries corresponding to a larger class of languages than the pattern languages. *Regular expressions with backreferencing* [4], as provided by *egrep* and Perl, for example, are in some sense a generalization of patterns in that substitutions of variables are restricted by regular expressions: the syntax $(e)\%X$, where e is a regular expression and X is a variable, binds a string $w \in L(e)$ to X. Subsequent uses of X in the expression then match w. It should be clear that we can easily extend the above construction of an ECRPQ for patterns to one that corresponds to a regular expression with backreferencing.

In fact, ECRPQs can match patterns, such as $a^n b^n c^n$, where $a, b, c \in \Sigma$ and $n \in \mathbb{N}$, that cannot be denoted by regular expressions with backreferencing, with the help of the equal length predicate:

$$\begin{aligned} Ans(x,y) \leftarrow\ & (x,\pi_1,z_1),(z_1,\pi_2,z_2),(z_2,\pi_3,y), \\ & a^*(\pi_1),b^*(\pi_2),c^*(\pi_3),\mathrm{el}(\pi_1,\pi_2),\mathrm{el}(\pi_2,\pi_3), \end{aligned}$$

where $el(\pi,\pi')$ is a shorthand for $(\bigcup_{a,b\in\Sigma}(a,b))^*(\pi,\pi')$.

Approximate matching and sequence alignment We treat approximate string matching and (biological) sequence alignment together because both are based on the notion of edit distance between strings. We consider the three edit operations of insertion, deletion and substitution, defined as follows. Let $s, s' \in \Sigma^*$. Applying an edit operation to s yielding s' can be modeled as a binary relation $\rightsquigarrow$ over Σ^* such that $x \rightsquigarrow y$ holds iff there exist $u, v \in \Sigma^*$, $a, b \in \Sigma$, with $a \neq b$, such that one of the following is satisfied:

$$\begin{array}{lll} x = uav, & y = ubv & \text{(substitution)} \\ x = uav, & y = uv & \text{(deletion)} \\ x = uv, & y = ubv & \text{(insertion)} \end{array}$$

Let $\overset{k}{\rightsquigarrow}$ stand for the composition of $\rightsquigarrow$ with itself k times. The *edit distance* $d_e(x,y)$ between x and y is the minimum number k of edit operations such that $x \overset{k}{\rightsquigarrow} y$.

We define a relation $D^{\leq k}$ between strings as follows: $(x,y) \in D^{\leq k}$ iff $d_e(x,y) \leq k$. This relation is regular (indeed, it is easy to see that it is accepted by a two-tape transducer, and the difference between the lengths of x and y is bounded by k; then it follows from the fact that rational relations of such bounded distance are regular [18]).

We now consider the use of edit distance in finding

string (or sequence) alignments. We can view an *alignment* of strings x and y over Σ at distance k as follows:

$$\begin{array}{cccccccc} x & = & x_0 & a_1 & x_1 & \cdots & a_k & x_k \\ y & = & y_0 & b_1 & y_1 & \cdots & b_k & y_k \end{array} \quad (3)$$

such that (i) $x_i, y_i \in \Sigma^*$ and $x_i = y_i$ for $i \in [0,k]$, and (ii) $a_i, b_i \in \Sigma \cup \{\epsilon\}$ and $a_i \neq b_i$, for $i \in [1,k]$. There is an alignment of x and y at distance k iff $(x,y) \in D^{\leq k}$. We call each pair (x_i, y_i) a *match* and each pair (a_i, b_i) a *mismatch* if $a_i, b_i \in \Sigma$ or a *gap* if a_i or b_i is ϵ. (If we allow that $a_i = b_i$, then we align the strings with distance *at most* k).

We have shown above that we can use an ECRPQ to determine whether there *exists* an alignment at distance k between two strings. However, we may also wish to return the actual gaps and mismatches to the user. For that, we assume that each node has an ϵ-labeled loop, and use an ECRPQ whose body is as follows

$$\bigwedge_{0\leq i\leq 2k}(x_i,\pi_i,x_{i+1}),\ \bigwedge_{0\leq i\leq 2k}(y_i,\rho_i,y_{i+1}),\ \bigwedge_{0\leq i\leq k}\pi_{2i}=\rho_{2i},\ \bigwedge_{1\leq i\leq k}R(\pi_{2i-1},\rho_{2i-1}),$$

where R is a finite language containing all pairs (a,b) in $\Sigma \cup \{\epsilon\}$ with $a \neq b$. The head of the query contains the variables π_{2i-1}, ρ_{2i-1}, for $1 \leq i \leq k$.

With the same approach, we can use ECRPQs to align not only pairs but arbitrary tuples of sequences. Multiple sequence alignment is used to find the shared evolutionary origins of biological sequences.

4. Query Evaluation

We now describe how ECRPQs can be evaluated. We need to take care of two aspects that distinguish ECRPQs from CRPQs: relations on paths, and path variables in the output. To deal with the former, we define a notion of *convolutions* of *db*-graphs and queries, that reduces the evaluation of ECRPQs to the evaluation of CRPQs. To deal with the latter, we produce an automaton construction that can represent both nodes and paths in the output.

Convolutions of graphs and queries We now present a construction that transforms a *db*-graph G and an ECRPQ Q into a *db*-graph G' and a CRPQ Q' with a single relational atom so that the evaluation of Q' over G' "coincides" (modulo a simple translation) with the evaluation of Q over G.

Let G be a Σ-labeled *db*-graph. By $G_\perp$ we denote the $\Sigma_\perp$-labeled *db*-graph obtained from G by adding a $\perp$-labeled loop to each node of G. We iteratively define G^m, the *m'th convolution* of G, as follows:

$$G^1 := G_\perp \ \text{ and } \ G^{m+1} = G_\perp \otimes G^m,$$

where $\otimes$ denotes the *product* of two *db*-graphs. We use the symbol $\otimes$ rather than $\times$ to indicate that this is not the standard product viewed as a graph/automaton over the same alphabet, but rather a graph over the product of alphabets. Formally, given a Σ_1-labeled *db*-graph $G_1 = (V_1, E_1)$ and a Σ_2-labeled *db*-graph $G_2 = (V_2, E_2)$, their *product* $G_1 \otimes G_2$ is the $(\Sigma_1 \times \Sigma_2)$-labeled *db*-graph $G = (V_1 \times V_2, E)$, where E contains edges $((v_1,v_2),(a,b),(v_1',v_2'))$, such that $(v_1,a,v_1') \in E_1$ and $(v_2,b,v_2') \in E_2$. Note that this makes G^m a $(\Sigma_\perp)^m$-labeled *db*-graph.

Consider an ECRPQ Q of the form:

$$Ans(\bar{z},\bar{\chi}) \leftarrow \bigwedge_{1\leq i\leq m}(x_{2i-1},\pi_i,x_{2i}),\ \bigwedge_{1\leq j\leq t}R_j(\bar{\pi}^j). \quad (4)$$

Note that the variables $x_1, \ldots, x_{2m}$ are not necessarily distinct. Let S_j $(1 \leq j \leq t)$ be the n_j-ary regular relation defined by R_j. We let L_Q be the regular expression over $(\Sigma_\perp)^m$ that represents the m-ary regular relation $S_Q = S_1(\bar{\pi}^1) \bowtie \ldots \bowtie S_t(\bar{\pi}^t)$. Note that S_Q is indeed regular since the class of regular relations is closed under intersection, projection, and product, and that relations of the form $\{\bar{s} \mid s_i = s_j\}$, which are necessary for defining joins, are regular as well.

The *convolution* of ECRPQ Q (4) is the CRPQ query Q_c defined as

$$Ans(y,y',\pi) \leftarrow (y,\pi,y'), L_Q(\pi). \quad (5)$$

Note that this is indeed a CRPQ over $(\Sigma_\perp)^m$-labeled *db*-graphs. Moreover, $Q_c(G^m)$, which consists of two m-tuples of nodes and a path in G^m, contains all the information we need to extract $Q(G)$; below, we show how to do this.

Let

$$\bar{\rho} \ = \ \bar{v}_0\bar{a}_0\bar{v}_1\bar{a}_1\bar{v}_2\cdots\bar{v}_{p-1}\bar{a}_{p-1}\bar{v}_p$$

be a path in G^m, where $\bar{v}_i = (v_i^1, \ldots, v_i^m)$ for each $i \leq p$ is a node in G^m, and $\bar{a}_i = (a_i^1, \ldots, a_i^m)$ for each $i \leq p-1$ is an element of $(\Sigma_\perp)^m$. Then, for each $j \leq m$, we let

$$\bar{\rho}(j) \ = \ v_0^j a_0^j v_1^j \cdots v_{p-1}^j a_{p-1}^j v_p^j$$

be a path in $G_\perp$. Notice that this is indeed a path in $G_\perp$ but not necessarily in G, as it may contain $\perp$-labeled loops. We then let $\bar{\rho}_s(j)$ stand for the path obtained from $\bar{\rho}(j)$ by eliminating all such loops $v \perp v$; this is now a path in G.

The output of $Q_c(G^m)$ consists of tuples of the form $(\bar{u}, \bar{u}', \bar{\rho})$, where $\bar{u} = (u_1, u_3, \ldots, u_{2m-1})$ and $\bar{u}' = (u_2, u_4, \ldots, u_{2m})$ are nodes in G^m and $\bar{\rho}$ is a path in G^m. We say that $(\bar{u}, \bar{u}')$ are Q-compatible if, whenever $x_i = x_j$ in Q, we have $u_i = u_j$, for all $i, j \leq 2m$. We now define the *Q-compatible output of Q_c on G^m* as the projection of the set

$$\left\{(\bar{u},\bar{u}',\bar{\rho}_s(1),\ldots,\bar{\rho}_s(m)) \;\middle|\; \begin{array}{l}(\bar{u},\bar{u}') \text{ is } Q\text{-compatible} \\ \text{and } (\bar{u},\bar{u}',\bar{\rho}) \in Q_c(G^m)\end{array}\right\}$$

onto the attributes that appear in the head of Q in (4).

That is, if x_i is among $\bar{z}$, we project onto u_i, and if π_j is among $\bar{\chi}$, we project onto $\bar{\rho}_s(j)$.

Theorem 4.1. *Let Q be an ECRPQ of the form (4) and G a db-graph. Then the Q-compatible output of the convolution CRPQ Q_c on G^m coincides with $Q(G)$.*

Representing paths in the answers Since ECRPQs can return paths, the answer to a query may be infinite (for example, if there is a cycle in the input graph, then we have infinitely many paths). In such cases we need to return a compact representation of the set of answers to an ECRPQ. It turns out that for each tuple of nodes $\bar{v}$, the set $\{\bar{\chi} \mid (\bar{v}, \bar{\chi}) \in Q(G)\}$ is a regular relation, and an automaton defining this relation can be constructed in time polynomial in the size of the input graph. We now present this construction.

Consider an ECRPQ Q of the form (2), i.e., $Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1\leq i\leq m}(x_i, \pi_i, y_i)$, $\bigwedge_{1\leq j\leq p} R_j(\bar{\pi}^j)$, a *db*-graph $G = (V, E)$, and a tuple $\bar{v}$ of nodes such that $|\bar{v}| = |\bar{z}|$. We let $Q(G, \bar{v})$ stand for the set $\{\bar{\rho} \mid (\bar{v}, \bar{\rho}) \in Q(G)\}$.

Let $|\bar{\chi}| = k$. We say that a path $\bar{\pi}$ in G^k *represents* a k-tuple of paths $(\rho_1, \ldots, \rho_k)$ in $Q(G, \bar{t})$ if $\bar{\pi}_s(j) = \rho_j$ for each $j \leq k$ and the label of $\bar{\pi}$ is precisely $[\lambda(\rho_1), \ldots, \lambda(\rho_k)]$. Recall that $\lambda(\cdot)$ stands for the label of a path; in particular, each $\lambda(\rho_j)$ is a string in Σ^*. Notice that such a path $\bar{\pi}$ is unique for the tuple $(\rho_1, \ldots, \rho_k)$, and in turn determines the tuple $(\rho_1, \ldots, \rho_k)$ uniquely.

Proposition 4.2. *For each ECRPQ Q with the head $Ans(z_1, \ldots, z_\ell, \chi_1, \ldots, \chi_k)$, db-graph $G = (V, E)$ and tuple $\bar{v} \in V^\ell$, one can construct, in polynomial time in $|E|$, an automaton $\mathcal{A}_Q^{(G,\bar{v})}$ over the alphabet $V^k \cup (\Sigma_\perp)^k$ that accepts precisely the representations of all the tuples of paths in $Q(G, \bar{v})$.*

5. Complexity of query evaluation

The reduction from ECRPQs to CRPQs gives us fairly easy upper bounds: one has to compute the convolution and evaluate a CRPQ over it. Using NLOGSPACE and NP bounds on the data and combined complexity of CRPQs, we conclude that the data complexity of ECRPQs is in PTIME, and their combined complexity is in EXPTIME. But can we do better?

It turns out that we can. For data complexity, we can lower the bound to NLOGSPACE: that is, the data complexity of CRPQs and ECRPQs is the same. For combined complexity, however, relations do make a difference: we show PSPACE-completeness of combined complexity. In the relational world, there are many techniques for lowering the NP combined complexity of conjunctive queries, typically by considering acyclic queries. This approach works for CRPQs, for which we show that acyclic queries can be evaluated in PTIME. However, when we move to ECRPQs, acyclicity does not lower the complexity. We then show that the techniques inspired by modeling infinite-state systems for verifying their temporal properties give us NP-completeness of combined complexity of classes of ECRPQs, matching the combined complexity of relational CQs.

5.1 Data complexity

If we fix a query Q over Σ, the problem we look at is the following:

PROBLEM:	ECRPQ-EVAL(Q)
INPUT:	A Σ-labeled *db*-graph G, a tuple $\bar{v}$ of nodes in G and a tuple $\bar{\rho}$ of paths in G.
QUESTION:	Does $(\bar{v}, \bar{\rho})$ belong to $Q(G)$?

The convolution technique, if applied carefully, gives us an NLOGSPACE upper bound. To evaluate the convolution query Q_c over G^m, we use an "on the fly" evaluation of the emptiness algorithm for the cross product of the automaton G^m, with a guessed assignment for the initial and final states, and the automaton $\mathcal{A}_Q$ that accepts the language L_Q of the convolution query. In the proof we still have to deal with some technical details (for instance, the presence of paths in the output for non-Boolean queries).

Theorem 5.1. *For each ECRPQ Q, the problem* ECRPQ-EVAL*(Q) is in* NLOGSPACE.

Since the problem can be NLOGSPACE-hard even for regular path queries that do not make use of path variables in the head [13], we also have a matching lower bound. Also note that when query Q is fixed, Proposition 4.2 tells us that there is a polynomial-size family of automata that represents the whole space of answers for Q over G.

5.2 Combined complexity

We now turn to the combined complexity, that is, query evaluation that takes both the *db*-graph and the query as the input:

PROBLEM:	ECRPQ-EVAL
INPUT:	A finite alphabet Σ, a Σ-labeled *db*-graph G, an ECRPQ Q over Σ, a tuple $\bar{v}$ of nodes in G and a tuple $\bar{\rho}$ of paths in G.
QUESTION:	Does $(\bar{v}, \bar{\rho})$ belong to $Q(G)$?

The problem CRPQ-EVAL is the restriction to when the query Q in the input is a CRPQ.

We start with the easier problem CRPQ-EVAL. It appears to be a folklore result (although we could not find it stated explicitly in the literature) that, without path variables in the head (i.e., the empty tuple $\bar{\rho}$) this problem is NP-complete. For the sake of completeness we present (in the full version) a proof of a slightly more general result that handles free path variables as well.

Proposition 5.2. CRPQ-EVAL *is* NP*-complete.*

However, adding regular relations to queries makes the query evaluation problem harder (at least under widely-held complexity theoretical assumptions). Notice that this is in stark contrast with what happens in the same case to the data complexity of the problem, where relations on paths do not increase the complexity.

Theorem 5.3. ECRPQ-EVAL *is* PSPACE*-complete. It remains* PSPACE*-hard even when restricted to Boolean ECRPQs.*

Note that the algorithm of Theorem 4.1 runs in single-exponential time; we give an on-the-fly construction of the automaton for computing the output that reduces the complexity to PSPACE. Hardness follows from encoding the regular expression intersection problem as an ECRPQ.

We now look at various approaches to lowering the complexity of query evaluation.

Acyclic queries It is, of course, a classical result of relational theory that acyclic conjunctive queries are tractable with respect to combined complexity. What if we require that the relational part of an (E)CRPQ be acyclic? Formally, we say that an ECRPQ or a CRPQ Q is acyclic if the graph H_Q of its relational part $\bigwedge_{1\leq i\leq m}(x_i, \pi_i, y_i)$, containing precisely the edges (x_i, y_i) for $i \leq m$, is acyclic.

The following result shows that the situation is drastically different for CRPQs and ECRPQs: the restriction works for the former but not for the latter. In fact, allowing only unary regular relations is precisely the boundary of tractability for the query evaluation problem restricted to acyclic ECRPQs.

Theorem 5.4.
- *The problem* CRPQ-EVAL *is in* PTIME*, if restricted to the class of acyclic CRPQs.*
- *The problem* ECRPQ-EVAL *is* PSPACE*-complete, even if restricted to the class of acyclic Boolean ECRPQs over a fixed alphabet Σ, that make use of regular relations of arity at most* 2*.*

For the first item, we show that the problem is reduced to evaluating acyclic CQs over the usual relational databases. The second item uses the reduction of Theorem 5.3, which requires Boolean acyclic queries and binary relations over a fixed alphabet.

Numerical representations of regular relations The idea comes from the field of verification of infinite-state systems, where regular languages are used to represents possible states of such systems (e.g., strings of states of an unbounded number of components of a system) and regular relations represent transitions between them [1]. While many problems related to verifying such systems are computationally hard or even undecidable, they become easier if an abstraction of a regular language presentation is used. Often such abstractions are in the form of definability in linear integer arithmetic, see, e.g., [35, 22, 34]. We shall now look at a similar idea of abstracting regular relations in connection with the ECRPQ evaluation problem.

We first look at relations that only talk about *length* of strings. More precisely, with each n-ary regular relation R, we associate a regular relation R_{len} defined as

$$\{(s_1, \ldots, s_n) \mid \exists (s'_1, \ldots, s'_n) \in R : \ |s_i| = |s'_i| \text{ for all } i\}.$$

Now, given an ECRPQ Q, we define Q_{len} as Q in which each relation R is replaced by R_{len}. This is still an ECRPQ due to the following:

Lemma 5.5. *If R is a regular relation, then so is R_{len}.*

It turns out that with this abstraction, we can lower the combined complexity to that of ordinary relational conjunctive queries.

Theorem 5.6. *The problem* ECRPQ-EVAL *for queries of the form Q_{len} is* NP*-complete.*

Note that in this case, the input contains a graph G and a query Q; the problem is to evaluate Q_{len} over G. The easier case, when the input already has regular expressions for relations R_{len}, has the same complexity.

The idea behind the proof of Theorem 5.6 is to view lengths of path as unary strings, thus representing them by unary automata (with some extra conditions, since we deal with relations on path lengths). We then use a polynomial-time algorithm that converts unary automata into a finite union of arithmetic progressions [12, 33] to encode the query evaluation problem as an existential sentence of Presburger arithmetic, which gives us the NP bound.

Another standard way of creating an abstraction of R is to count the numbers of occurrences of symbols; this, however, leads to non-regular languages and relations (e.g., if we only count numbers of occurrences of letters, strings from $(ab)^*$ will be transformed into strings in which the number of a's equals the number of b's). Nonetheless, we can prove an NP upper bound for such nonregular relations; they will be considered in the next section when we look at extensions of ECRPQs.

Repetition of path variables In the definitions of CRPQs and ECRPQs, repetition of path variables is allowed in neither the relational parts nor the regular languages/relations. For instance, we cannot write $(x, \pi, y), (x', \pi, y')$, nor can we write $R_1(\bar{\omega}), R_2(\bar{\omega})$. We

refer to these as relational repetitions and regular repetitions, respectively. What happens if these are allowed? It turns out that the complexity of ECRPQs is not affected, but the combined complexity of CRPQs jumps to that of ECRPQs, no matter what kind of repetition is allowed.

Proposition 5.7. *The problem* ECRPQ-EVAL *remains in* PSPACE *for ECRPQs with any kind of repetition, while the problem* CRPQ-EVAL *becomes* PSPACE*-complete even for Boolean acyclic CRPQs with either relational or regular repetitions.*

6. Query Containment

The task of checking query containment is crucial for problems such as query optimization and data integration. The problem of query containment for CRPQs was first introduced in [16] which showed an EXPSPACE upper bound. A matching lower bound was then shown in [10]. Here we study the problem of query containment for ECRPQs, i.e., checking for two ECRPQs Q and Q' over Σ, whether $Q(G) \subseteq Q'(G)$ for every Σ-labeled *db*-graph G.

We have indicated in Section 3 how string patterns (with variables) can be coded by ECRPQs. Combining this with a recent result on the undecidability of pattern containment [17], we can prove the following.

Theorem 6.1. *There exists a fixed finite alphabet Σ, such that the containment problem for ECRPQs over Σ is undecidable.*

We can recover decidability if one of the queries is a CRPQ. The problem of containment of an ECRPQ in a CRPQ is the problem of checking whether $Q(G) \subseteq Q'(G)$ for every Σ-labeled *db*-graph G, where Q is an ECRPQ and Q' is a CRPQ over Σ. We can adapt proof techniques from [10] to show that the above problem has the same complexity as CRPQ containment.

Theorem 6.2. *The problem of checking containment of an ECRQP in a CRPQ is* EXPSPACE*-complete.*

It is still open whether the other version of the problem – the containment of a CRPQ in an ECRPQ – is decidable.

7. Extensions

We now look at various ways of going beyond the class of ECRPQs. First, we consider an analog of relational calculus by adding negation and arbitrary quantification to the language. After that, we look at conjunctive queries with non-regular relations; we handle linear constraints on lengths of paths, and Parikh-image constraints.

7.1 Adding negation

We now investigate the query evaluation problem for the extension of ECRPQs with negation. Formally, we define the language ECRPQ$^\neg$ over alphabet Σ as the set of formulas described by the following grammar:

$$\begin{array}{rcl} \text{atom} & := & \pi_1 = \pi_2 \mid x = y \mid (x, \pi, y) \mid R(\pi_1, \ldots, \pi_n) \\ \phi, \psi & := & \text{atom} \mid \neg\phi \mid \phi \vee \psi \mid \exists x \phi \mid \exists \pi \phi \end{array}$$

Here x, y range over the set of node variables, $\pi, \pi_1, \ldots$ range over the set of path variables, and R ranges over the set of regular expressions over alphabets $(\Sigma_\perp)^n$ $(n > 0)$ that represent n-ary regular relations over Σ. The language CRPQ$^\neg$ is defined as the restriction of ECRPQ$^\neg$ to formulas that only make use of regular languages. The notions of free and bound variables are standard; we write $\phi(\bar{x}, \bar{\pi})$ to list free node and path variables explicitly.

The semantics of ECRPQ$^\neg$ is defined in the standard way. Given a *db*-graph $G = (V, E)$, a mapping σ from the set of free node variables of ϕ into V, and a mapping μ from the set of free path variables of ϕ into the set of paths in G, the notion $(G, \sigma, \mu) \models \phi$ is defined just as for ECRPQs with the following additional rules:

- the Boolean connectives $\neg$ and $\vee$ have the standard semantics;
- $(G, \sigma, \mu) \models \exists x \phi$ iff there exists $v \in V$ such that $(G, \sigma_{x \to v}, \mu) \models \phi$, where $\sigma_{x \to v}$ extends the assignment σ by letting $\sigma(x) = v$;
- $(G, \sigma, \mu) \models \exists \pi \phi$ iff there exists a path ρ such that $(G, \sigma, \mu_{\pi \to \rho}) \models \phi$, where $\mu_{\pi \to \rho}$ extends the assignment μ by letting $\mu(\pi) = \rho$.

Given a *db*-graph G and an ECRPQ$^\neg$ formula $\phi(\bar{x}, \bar{\pi})$, we let $\phi(G)$ be the set of tuples $(\bar{v}, \bar{\rho})$ such that $(G, \sigma, \mu) \models \phi$, where $\sigma(\bar{x}) = \bar{v}$ and $\mu(\bar{\pi}) = \bar{\rho}$.

Notice that ECRPQ$^\neg$ and CRPQ$^\neg$ express nontrivial properties of *db*-graphs that are not expressible by means of ECRPQs. For instance, the query $\neg\exists\pi\,((x, \pi, y) \wedge L(\pi))$ defines the set of all pairs (a, b) of nodes such that no path between them is labeled by a string from language L.

Combined complexity The problems ECRPQ$^\neg$-EVAL and CRPQ$^\neg$-EVAL are defined exactly as the query evaluation problems in Section 5.2 except that the input query is from the extended language. Again we see a significant difference between regular languages and regular relations in queries. The complexity jumps in both cases, but while CRPQ$^\neg$-EVAL can be solved in single-exponential time, ECRPQ$^\neg$ queries cannot be evaluated in time bounded by a fixed stack of exponents.

Theorem 7.1.
- *The problem* CRPQ$^\neg$-EVAL *is* PSPACE*-complete.*
- *The problem* ECRPQ$^\neg$-EVAL *is decidable, but non-elementary.*

For the first item, we view the problem of query evaluation as the problem of evaluating FO sentences over a certain infinite automatic structure, essentially formed by paths in a *db*-graph. We use a game argument to show that such evaluation only needs quantification over a small finite part of the structure, whose elements are of size at most polynomial in the size of the input. This leads to a PSPACE bound.

For the second problem, decidability follows from extending our automaton construction for obtaining query answers, but at the cost of the non-elementary blow up, since each negation leads to an exponential increase in the size of the automaton. We then use bounds on model-checking over automatic structures [34] to conclude that this blow up is unavoidable.

Data complexity We now turn to data complexity, with the *db*-graph as the sole input. That is, we look at problems CRPQ$^\neg$-EVAL(ϕ) and ECRPQ$^\neg$-EVAL(ϕ), for each query ϕ. Again we see a significant gap between allowing regular languages and relations. In the former case, the complexity matches that of the CRPQs, while in the latter case, the problem can be intractable. Establishing exact bounds (in particular, whether the complexity of ECRPQ$^\neg$-EVAL(ϕ) could be non-elementary hinges upon some problems related to query evaluation over automatic structures which are currently open (see, e.g., [8, 34]).

Proposition 7.2.
- *For each* CRPQ$^\neg$ *formula* ϕ*, the problem* CRPQ$^\neg$-EVAL*(*ϕ*) is in* NLOGSPACE.
- *There exists a query* ϕ *in* ECRPQ$^\neg$ *such that the problem* ECRPQ$^\neg$-EVAL*(*ϕ*) is* PSPACE*-hard.*

We know of course that the containment problem is undecidable for ECRPQs, and thus for ECRPQ$^\neg$ queries. We remark that even a simpler satisfiability problem, asking whether for a Boolean ECRPQ$^\neg$ query ϕ there is a *db*-graph G such that $\phi(G) = \texttt{true}$, is undecidable. This is because the finite satisfiability problem for arbitrary FO formulas over binary predicates can easily (and effectively) be encoded into the satisfiability problem for ECRPQ$^\neg$ formulas.

7.2 Adding non-regular relations

A well-known class viewed as a natural extension of regular relations is the class of *rational relations*. Binary rational relations can be viewed as sets of pairs (s, s') of strings over Σ such that s' is a possible output of a nondeterministic *transducer* on s. However, a standard PCP reduction shows the following:

Proposition 7.3. *If rational relations are allowed in place of regular relations in ECRPQs, then the query evaluation problem becomes undecidable.*

Hence, we need to work with weaker extensions. We now look at two such examples.

Linear constraints on lengths of paths As we mentioned earlier, sometimes it is desirable to compare the length of paths. We define a class of *CRPQs with length comparisons* as

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1\le i\le m} (x_i, \pi_i, y_i),\ \bigwedge_{1\le j\le t} L_j(\omega_j),\ \mathbf{A}\bar{\ell} \ge \mathbf{b}\,,$$

where $Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1\le i\le m}(x_i, \pi_i, y_i), \bigwedge_{1\le j\le t} L_j(\omega_j)$ is a CRPQ as in (1) (note that we allow path variables in the head), and

- $\mathbf{A}$ is a $k \times m$ matrix over $\mathbb{Z}$, for some $k > 0$;
- $\mathbf{b}$ is a vector in $\mathbb{Z}^k$; and
- $\bar{\ell} = (\ell_1, \ldots, \ell_m)$.

The semantics is extended as follows: each ℓ_i is interpreted as the length of the path π_i, for $i \le m$. The last clause of the query is true if $\mathbf{A}\ell \ge \mathbf{b}$ under this interpretation. That is, we extend CRPQs with the ability to add $k > 0$ linear constraints on lengths of paths.

Theorem 7.4. *The combined complexity of CRPQs with length comparisons is* NP*-complete, and their data complexity is in* PTIME.

Thus, from the point of view of overall complexity, adding length comparisons is free, as it does not increase the combined complexity of CRPQs. Whether this remains true for ECRPQs, and whether the PTIME bound can be lowered, remains open.

The proof of Theorem 7.4 again uses unary automata to code lengths of paths satisfying the subgoals of the query. We convert these into unions of arithmetic progressions [12, 33] and then combine them with linear constraints in a single Presburger formula which can be evaluated in NP. Analyzing this algorithm for a fixed query and using polynomiality of integer linear programming in fixed dimension [28] we get the PTIME bound of data complexity.

Counting occurrences of letters We now look at another way for abstracting regular relations. While taking us outside the class of regular relations, it actually lowers the complexity of query evaluation to match the complexity of relational conjunctive queries.

Recall that the abstraction we used before was based on just looking at lengths of strings in tuples of a relation. We now refine this (in fact, in two ways), and look at numbers of occurrences of each alphabet symbol, i.e., at Parikh images of strings. Recall that the *Parikh image* $\mathrm{par}(s)$ for a string s over the alphabet $\{a_1, \ldots, a_k\}$ is a tuple $(n_1, \ldots, n_k) \in \mathbb{N}^k$, where n_i is the number of occurrences of a_i in s, for each $i \le k$. Now let R be a regular n-ary relation over Σ. Similarly to R_{len}, we define a relation R_{par} as

$$\left\{ (s_1, \ldots, s_n) \;\middle|\; \exists (s'_1, \ldots, s'_n) \in R : \begin{array}{l} \mathrm{par}(s_i) = \mathrm{par}(s'_i) \\ \text{for all } i \le n \end{array} \right\}.$$

Complexity	CQs	CRPQ	ECRPQ	Acyclic CRPQ	Acyclic ECRPQ	ECRPQ Q_{len}
data	AC^0	NL-complete	NL-complete	NL-complete	NL-complete	NL-complete
combined	NP-complete	NP-complete	PSPACE-complete	PTIME	PSPACE-complete	NP-complete

(a) CQs, CRPQs, ECRPQs, and subclasses

Complexity	with repetitions CRPQ	with repetitions ECRPQ	with negation CRPQ	with negation ECRPQ	CRPQ with linear constraints	Q°_{par} and Q_{par}, fixed arity
data	NL-complete	NL-complete	NL-complete	PSPACE-hard	PTIME	PTIME
combined	PSPACE-complete	PSPACE-complete	PSPACE-complete	nonelementary	NP-complete	NP-complete

(b) Extensions of CRPQs and ECRPQs

Figure 1: Summary of complexity results for classes of queries

We also define a relation R°_{par} as the set of tuples $(s_1, \ldots, s_n)$ such that for each $i \leq n$, there is a tuple in R whose ith component s'_i satisfies $\text{par}(s_i) = \text{par}(s')$. Note that $R \subseteq R_{\text{par}} \subseteq R^{\circ}_{\text{par}}$. These relations are not necessarily regular: e.g., $((ab)^*)_{\text{par}}$ is the set of strings with equal numbers of a's and b's. Nonetheless, these abstractions can be useful if we only care about occurrence numbers (like, for example, in dtds over unordered trees), and need to lower the complexity of query evaluation.

For an ECRPQ Q, we define Q_{par} and Q°_{par} as Q in which all occurrences of regular relations are replaced by R_{par} or R°_{par}, respectively. By the remark above, these are not necessarily ECRPQs.

Proposition 7.5. *The combined complexity for queries of the form Q°_{par} is* NP*-complete, as is the complexity of queries Q_{par} when the arity of relations is fixed. In both cases the data complexity is in* PTIME*.*

The proof again uses the main constructions of the proof of Theorem 5.6 together with conversions of Parikh images into existential Presburger formulae [35], to encode query evaluation as satisfiability of existential Presburger formulae. For data complexity, the analysis of the resulting Presburger formula for a fixed query shows that its satisfiability reduces to solving a fixed-dimension instance of integer linear programming, which is known to be polynomial [28].

8. Summary and Future Work

The tables in Figure 1 give the summary of the complexity results; they also, include, for comparison, known results on CQs and CRPQs in the first two columns of (a). We use the abbreviation NL for NLOGSPACE; note that all NLOGSPACE bounds actually give NLOGSPACE-completeness since every version of path queries can express reachability. In addition to these complexity results, our technical results also include algorithms for query evaluation and representation of paths in the output and results on query containment.

Several basic problems related to ECRPQs are still open. We do not known whether checking the containment of a CRPQ in an ECRPQ is decidable, nor do we know if it is decidable to check whether a given ECRPQ is equivalent to a CRPQ. These are important problems for understanding the computability of view-based query rewriting of ECRPQs, in the line of [11], and for query optimization.

We also would like to extend results on queries with linear constraints on lengths of paths, and find practical algorithms for classes of queries whose combined complexity matches that of relational conjunctive queries. Further pursuing the relational analogy, we would like to investigate the parameterized complexity of classes of ECRPQs. While we have seen that the standard acyclicity restriction does not help us, it is natural to look for conditions that will guarantee fixed-parameter tractability.

Acknowledgment This work started when the first two authors visited the last author at Birkbeck, University of London, funded by a Royal Society International Joint Project. We also gratefully acknowledge support by FONDECYT grant 11080011 (Barceló), EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599 (Libkin). Discussions with Marcelo Arenas, Claudio Gutiérrez, Alex Poulovassilis and Anthony Widjaja To during different stages of this project have been of great help.

9. References

[1] P. A. Abdulla, B. Jonsson, M. Nilsson, M. Saksena. A survey of regular model checking. In *CONCUR'04*, pages 35–48.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Wiener. The LOREL query language for semistructured data. *Int. J. Digit. Libr.*, 1(1):68–88, 1997.

[3] S. Abiteboul, P. Buneman, D. Suciu. *Data on the web: From relations to semistructured data and XML*. Morgan Kauffman, 1999.

[4] A. V. Aho. Algorithms for finding patterns in strings. In *Handbook of TCS*, Vol. A, pages 255–300, 1990.

[5] K. Anyanwu, A. P. Sheth. ρ-Queries: enabling querying for semantic associations on the semantic web. In *WWW'03*, pages 690–699.

[6] C. L. Barrett, R. Jacob, M. V. Marathe. Formal-language-constrained path problems. *SIAM J. Comput.*, 30(3):809–837, 2000.

[7] M. Benedikt, L. Libkin, T. Schwentick, L. Segoufin. Definable relations and first-order query languages over strings. *JACM*, 50(5):694–751, 2003.

[8] A. Blumensath, E. Grädel. Automatic structures. In *LICS'00*, pages 51–62.

[9] V. Bruyère, G. Hansel, C. Michaux, R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Society*, 1:191–238, 1994.

[10] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR'00*, pages 176–185.

[11] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3):443–465, 2002.

[12] M. Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(2):149–158, 1986.

[13] M. P. Consens, A. O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS'90*, pages 404–416.

[14] A. Deutsch, V. Tannen. Optimization properties for classes of conjunctive regular path queries. In *DBPL'01*, pages 21–39.

[15] C. Elgot, J. Mezei. On relations defined by generalized finite automata. *IBM Journal Research Develop*, 9(1):47-68, 1965.

[16] D. Florescu, A. Levy, D. Suciu. Query containment for conjunctive queries with regular expressions. In *PODS'98*, pages 139–148.

[17] D. D. Freydenberger, D. Reidenbach. Bad news on decision problems for patterns. *Information and Computation*, 208(1):83–96, 2010.

[18] C. Frougny, J. Sakarovitch. Rational relations with bounded delay. In *STACS'91*, pages 50–63.

[19] G. Grahne, A. Thomo. Query answering and containment for regular path queries under distortions. In *FoIKS'04*, pages 98–115.

[20] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[21] D. A. Holland, U. Braun, D. Maclean, K.-K. Muniswamy-Reddy, M. I. Seltzer. Choosing a data model and query language for provenance. In *Int. Provenance and Annotation Workshop*, 2008.

[22] O. Ibarra, J. Su, Z. Dang, T. Bultan, R. Kemmerer. Counter machines and verification problems. *Theor. Comput. Sci.*, 289(1):165–189, 2002.

[23] Y. Kanza, Y. Sagiv. Flexible queries over semistructured data. In *PODS'01*, pages 40–51.

[24] K. Kochut, M. Janik. SPARQLeR: Extended SPARQL for semantic association discovery. In *ESWC'07*, pages 145–159.

[25] D. Kozen. Lower bounds for natural proof systems. In *FOCS'77*, pages 254-266.

[26] W.-J. Lee, L. Raschid, P. Srinivasan, N. Shah, D. L. Rubin, N. F. Noy. Using annotations from controlled vocabularies to find meaningful associations. In *Proc. Workshop on Data Integr. in Life Sciences*, pages 247–263, 2007.

[27] J. Lehmann, J. Schüppel, and S. Auer. Discovering unknown connections—the DBpedia relationship finder. In *Conf. on Social Semantic Web*, pages 99–110, 2007.

[28] H. W. Lenstra. Integer programming in a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.

[29] A. O. Mendelzon, P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995.

[30] T. Milo, D. Suciu. Index structures for path expressions. In *ICDT'99*, pages 277-295.

[31] C. H. Papadimitriou. On the complexity of integer programming. *JACM*, 28(4):765–768, 1981.

[32] A. Sheth et al. Semantic association identification and knowledge discovery for national security applications. *J. Database Management*, 16(1):33–53, 2005.

[33] A.W. To. Unary finite automata vs. arithmetic progressions. *IPL*, 109(17):1010–1014, 2009.

[34] A.W. To. Model checking FO(R) over one-counter processes and beyond. In *CSL'09*, pages 485–499.

[35] K. N. Verma, H. Seidl, T. Schwentick. On the complexity of equational Horn clauses. In *CADE'05*, pages 337–352.

[36] G. Weikum, G. Kasneci, M. Ramanath, and F. Suchanek. Database and information-retrieval methods for knowledge discovery. *Commun. ACM*, 52(4):56–64, 2009.