Answering Regular Path Queries Using Views

Diego Calvanese[1], Giuseppe De Giacomo[1], Maurizio Lenzerini[1], Moshe Y. Vardi[2]

[1] *Dipartimento di Informatica e Sistemistica*
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it

[2] *Department of Computer Science*
Rice University, P.O. Box 1892
Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu

Abstract

Query answering using views amounts to computing the answer to a query having information only on the extension of a set of views. This problem is relevant in several fields, such as information integration, data warehousing, query optimization, mobile computing, and maintaining physical data independence. We address query answering using views in a context where queries and views are regular path queries, i.e., regular expressions that denote the pairs of objects in the database connected by a matching path. Regular path queries are the basic query mechanism when the database is conceived as a graph, such as in semistructured data and data on the web. We study algorithms for answering regular path queries using views under different assumptions, namely, closed and open domain, and sound, complete, and exact information on view extensions. We characterize data, expression, and combined complexity of the problem, showing that the proposed algorithms are essentially optimal. Our results are the first to exhibit decidability in cases where the language for expressing the query and the views allows for recursion.

1. Introduction

Query answering using views amounts to computing the answer to a query having information only on the extension of a set of views. This problem is relevant in several fields, such as information integration [31], data warehousing [33], query optimization [11], mobile computing [5], and maintaining physical data independence [30].

Data integration is the setting that has put the strongest emphasis on query answering using views: a typical integration process results in a set of precomputed views, and the query evaluation mechanism can only rely on such views in order to derive correct answers to queries. Two approaches to data integration have been investigated, called virtual and materialized. In the virtual approach, the precomputed views represent the data sources that are integrated, whereas in the materialized approach (generally adopted in data warehousing), the precomputed views represent the result of the integration activity carried out over the sources. In both cases, the problem of answering queries using views is crucial.

When integrating data from many autonomous sources, each with differing modeling features and assumptions, e.g., data sources on the web, it is convenient to resort to modeling mechanisms that are flexible and adaptable. This has raised interest in the management of semistructured data, which are data that do not fit into rigid, predefined schemas, and are best described by graph-based data models [7, 1, 17].

Methods for extracting information from semistructured data necessarily incorporate special querying mechanisms that are not common in traditional database systems. One such basic mechanism is the one that retrieves all pairs of nodes in the graph connected by a path conforming to a regular expression (regular path queries) [9, 3]. Observe that regular expressions provide a (limited) form of recursion, which is used in regular path queries to navigate the graph database.

In this paper we address the problem of query answering using views when both the query and the views are regular path queries. Our goal is to devise algorithms and characterize the computational complexity of the problem under different assumptions. This represents a fundamental step towards solving the problem of query answering using views for full-fledged query languages over semistructured data and data on the web [9, 3, 15, 13].

The assumptions that we consider are on the information available on the domain, and on the information on the view extensions [2, 20].

The *closed domain assumption* states that the database contains exactly the objects stored in the views. In other words, although we do not know the exact form of the database, we know the set of objects stored in it. On the contrary, under the *open domain assumption* the database

may contain other objects besides those stored in the views.

A view is *exact* if its extension is exactly the set of objects in the database that satisfy its definition. A view is *sound* if its extension is a subset of the objects that satisfy its definition. In other words, when a view is sound we know a subset of the pairs of objects that satisfy the view, but we cannot exclude that other pairs of objects satisfy the view as well. The case of complete view is the dual one: a view is *complete*, if its extension is a superset of the pairs of objects in the database that satisfy its definition. Observe that an exact view is one that is both sound and complete. When answering a query using views, sound views are used to infer pairs of objects that are in the answer set of the query, while complete views are used to infer pairs of objects that are not in the answer set of the query.

As pointed out in [20], in data integration, a sound view corresponds to a data source that is known to produce only (not necessarily all) the answers to the query associated to the view. On the other hand, a complete view models a source that is known to produce all answers to the associated query, and maybe more. Finally, an exact view models a source that is known to produce exactly the answers to the associated query.

The framework we consider in the paper allows the specification of which assumption to adopt for the domain of the database, and of which one to adopt for each of the available views. Within this framework, we present the following results:

- We provide algorithms for query answering using views thus showing that all cases are decidable. We study the data, expression, and combined complexity of the algorithms.
- We characterize the lower bounds of the problem, and we show that such lower bounds match the upper bounds provided by the algorithms. In particular, we show that answering regular path queries using views is coNP-complete with respect to data complexity in all cases. With respect to expression (and hence combined) complexity, the problem is coNP-complete under the closed domain assumption, and PSPACE-complete under the open domain assumption.

Our investigation is similar in spirit to the one presented in [2], where the decidability and the data complexity of the problem is studied when the views and the queries are expressed in terms of various languages (conjunctive queries, Datalog, first-order queries, etc.). The results in [2] show that answering recursive (Datalog) queries using recursive views is undecidable in general. The results presented in this paper are the first to exhibit decidability in a case where the language for expressing the query and the views allows for recursion.

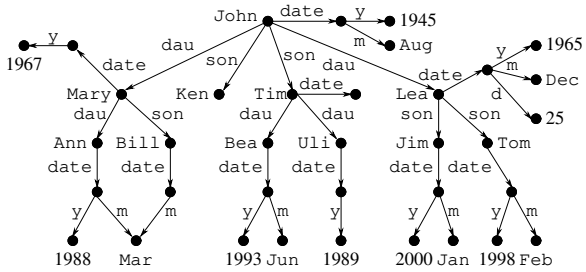

Figure 1. A database

The paper is organized as follows. Section 2 sets the framework in which we study the problem of query answering using views. Section 3 provides an overview of the results, and a comparison with related work. Section 4 investigates the problem in the case of the closed domain assumption. Section 5 deals with answering queries using sound views under the open domain assumption. Section 6 presents the results for the case of open domain assumption and arbitrary views. Section 7 investigates the connection between answering queries using views and query rewriting using views. Finally, Section 8 concludes the paper.

2. Framework

We consider a setting in which databases are expressed in terms of edge-labeled graphs, and queries ask for pairs of nodes connected by a specified path. This setting is typical in semistructured data, where all data models share the characteristic that data are organized in a labeled graph, where the nodes represent objects, and the edges represent links between objects [26, 8, 7, 1, 17].

2.1. Regular path queries

Formally, we consider a *database* as an edge labeled graph $DB = (\mathcal{D}, \mathcal{E})$, where $\mathcal{D}$ is a set of nodes (called the *domain*) that represent the objects of DB, and $\mathcal{E} = \{r_e \mid e \in \Sigma\}$ is a set of binary relations corresponding to the edges of the graph labeled by elements from an alphabet Σ. Such edges represent links between objects labeled by attribute names. We denote an edge from node x to node y labeled by r, i.e., $(x, y) \in r$, with $x \xrightarrow{r} y$.

Example 1 We show in Figure 1 an example of a database with information on a set of people, their sons and daughters, and their date of birth. ∎

As query mechanism we consider *regular path queries* (simply *queries* in the following), which are the basic constituents of full-fledged query languages over semistructured data [9, 1, 16, 24, 13]. Such queries denote all the

paths corresponding to words of a specified regular language over the alphabet Σ, and hence are expressed by means of regular expressions or finite automata [10].

Definition 2 The *answer set to a query Q over a database DB* is $ans(Q, DB) = \{(x, y) \mid$ there is a path $x \xrightarrow{r_1} \cdots \xrightarrow{r_n} y$ in DB s.t. $r_1 \cdots r_n \in L(Q)\}$, where $L(Q)$ is the regular language defined by Q. ∎

Example 3 Refer to the database in Figure 1, and consider the query $(\texttt{son} + \texttt{dau})^* \cdot \texttt{dau} \cdot \texttt{date} \cdot \texttt{m}$, asking for the pairs (x, y) such that y is the month of birth of a female descendent of x. It is easy to see that the answer set to the query contains e.g., $(\texttt{John}, \texttt{Jun})$ and $(\texttt{John}, \texttt{Dec})$. ∎

2.2. Query answering using views

We now introduce formally the problem of query answering using views. As pointed out in [2, 20, 22], this problem comes in different forms, depending on various assumptions about how accurate is the knowledge on both the objects of the database, and the pairs satisfying the views.

Consider a database that is accessible only through a set of views $V_1, \ldots, V_k$, and suppose we want to answer a regular path query only on the basis of our knowledge on the views. Specifically, associated to each view V_i we have:

- its definition $def(V_i)$ in terms of a regular path query over the alphabet Σ;
- information about its extension in terms of
  - a set $ext(V_i)$ of pairs of objects[1];
  - a specification $as(V_i)$ of which *assumption* to adopt for the view V_i in interpreting $ext(V_i)$ with respect to the answer set of $def(V_i)$.

We consider the following three assumptions on views [2, 20]:

- *Sound View Assumption* (SVA). We say that a view V_i is *sound* (satisfies SVA) with respect to a database DB, if $ext(V_i) \subseteq ans(def(V_i), DB)$. This means, that from the fact that a pair (a, b) is in $ext(V_i)$ we can conclude that (a, b) is in $ans(def(V_i), DB)$. However, if (a, b) is not in $ext(V_i)$ we cannot conclude that (a, b) is not in $ans(def(V_i), DB)$.
- *Complete View Assumption* (CVA). We say that a view V_i is *complete*, (satisfies CVA) with respect to a database DB, if $ext(V_i) \supseteq ans(def(V_i), DB)$. This means, that from the fact that a pair (a, b) is in $ext(V_i)$ we cannot conclude that (a, b) is in $ans(def(V_i), DB)$. On the other hand, if (a, b) is not in $ext(V_i)$ we can conclude that (a, b) is not in $ans(def(V_i), DB)$.
- *Exact View Assumption* (EVA). We say that a view V_i is *exact* (satisfies EVA) with respect to a database DB, if $ext(V_i) = ans(def(V_i), DB)$. This means, that we know that the extension of the view is exactly the set of pairs of objects that satisfy the view.

We say that a database DB is *consistent with a view V_i* if V_i satisfies the assumption $as(V_i)$ with respect to DB.

Example 4 A possible set of views for the database of Figure 1 is $\{V_1, V_2, V_3\}$ where:

$$\begin{array}{lcl} def(V_1) & = & (\texttt{son} + \texttt{dau})^* \cdot \texttt{dau} \cdot \texttt{date} \cdot \texttt{m} \\ def(V_2) & = & \texttt{dau} \\ def(V_3) & = & \texttt{son} + \texttt{dau} \end{array}$$

Suppose that the extension $ext(V_1)$ is $\{(\texttt{John}, \texttt{Jun})\}$, the extension $ext(V_2)$ is $\{(\texttt{John}, \texttt{Mary}), (\texttt{Mary}, \texttt{Ann}), (\texttt{Tim}, \texttt{Bea}), (\texttt{Tim}, \texttt{Uli}), (\texttt{John}, \texttt{Lea}), (\texttt{John}, \texttt{Tim})\}$, and the extension $ext(V_3)$ is the set of nodes connected by $\texttt{son}$ or $\texttt{dau}$. Then V_1 is sound, V_2 is complete, and V_3 is exact. Hence, if $as(V_1) = \text{SVA}$, $as(V_2) = \text{CVA}$, $as(V_3) = \text{EVA}$, then the database of Figure 1 is consistent with V_1, V_2, and V_3. ∎

In the following, we denote by $\mathcal{D}_V$ the set of objects appearing in $ext(V_1) \cup \cdots \cup ext(V_k)$. With respect to the information available on the objects in the database, we further distinguish between:

- *Closed Domain Assumption* (CDA). The exact set of objects in the database coincides with the set of objects that appear in the view extensions. We say that a database $(\mathcal{D}, \mathcal{E})$ is *consistent with $\mathcal{D}_V$ under CDA* if $\mathcal{D}_V$ is equal to $\mathcal{D}$.
- *Open Domain Assumption* (ODA). Only a subset of the objects in the database appears in the view extensions. We say that a database $(\mathcal{D}, \mathcal{E})$ is *consistent with $\mathcal{D}_V$ under ODA* if $\mathcal{D}_V$ is a subset of $\mathcal{D}$.

We are now ready to define the problem of answering queries using views.

Definition 5 The problem of *answering queries using views under CDA (resp., ODA)* is the following: Given

- $def(V_i)$, $ext(V_i)$, and $as(V_i)$, for $1 \leq i \leq k$
- a query Q
- a pair of objects $c, d \in \mathcal{D}$

[1] We assume that objects are represented by constants, and we adopt the *unique name assumption* [28], i.e., different constants denote different objects and therefore different nodes.

decide whether $(c, d) \in ans(Q)$, i.e., decide whether $(c, d) \in ans(Q, DB)$, for every DB that is consistent with $\mathcal{D}_V$ under CDA (resp., ODA) and that is consistent with $V_1, \ldots, V_k$. ∎

The complexity of the problem can be measured in three different ways [32]:

- *Data complexity*: as a function of the size of $ext(V_1) \cup \cdots \cup ext(V_k)$.
- *Expression complexity*: as a function of the size of Q and of the expressions $def(V_1), \ldots, def(V_k)$.
- *Combined complexity*: as a function of the size of both $ext(V_1) \cup \cdots \cup ext(V_k)$ and the expressions $Q, def(V_1), \ldots, def(V_k)$.

In the following, when we say that a DB is consistent with a set of views, we implicitly mean that the DB is also consistent with $\mathcal{D}_V$ under the current domain assumption.

2.3. Relationships between the different assumptions

SVA and EVA are inherently different assumptions. To see this, it is sufficient to note that if we adopt EVA for some of the views, then there is the possibility that there exists no database at all which is consistent with the views. This cannot happen in the case where all views are sound.

Example 6 Consider views V_1 and V_2 such that

$$def(V_1) = R \quad ext(V_1) = \{(a, c)\} \quad as(V_1) = \text{SVA}$$
$$def(V_2) = R^* \quad ext(V_2) = \{(a, b)\} \quad as(V_2) = \text{EVA}$$

Obviously, from the extension of V_1 we can conclude that (a, c) should also appear in V_2. Since V_2 is assumed to be exact, no database exists which is consistent with the views. ∎

On the other hand, complete views can be reformulated in terms of exact views. Indeed, by exploiting union in our query language, given an instance of the problem of query answering using views, we can always transform it to a new instance with only sound and exact views, and such that the solutions of the two instances are the same. Suppose we want to check whether $(c, d) \in ans(Q)$ under either CDA or ODA, given the views $V_1, \ldots, V_k$, and suppose that $as(V_i) = \text{CVA}$. Introduce in Σ a new relation symbol R_{new} that does not appear in $Q, V_1, \ldots, V_k$, and replace V_i by V_i' with $def(V_i') = def(V_i) + R_{new}$, $ext(V_i') = ext(V_i)$, and $as(V_i') = \text{EVA}$. It is easy to see that the new instance of the problem has the same solution as the original one. For this reason, in what follows, we concentrate on sound and exact views only. Note that we cannot apply similar arguments in order to reduce sound views to exact views, because our query language lacks intersection[2].

With respect to the relationship between CDA and ODA we observe that CDA imposes more restrictions than ODA on the databases that are consistent with the views. Namely, under CDA, a database that is consistent with the views, must contain only the objects that are in the view extensions. Instead, under ODA, a database that is consistent with the views, may contain objects that are not in the view extensions. Hence in verifying that a given pair of objects is in the answer set of a query, under CDA, we must take into account only databases containing exactly the objects in the views, while, under ODA, we must take into account also databases which contain additional objects. The following example illustrates such a difference.

Example 7 Suppose $def(V) = R_1 \cdot R_2$, $ext(V) = \{(a, b)\}$, and we want to check whether $(a, b) \in ans(R_1 + R_2)$. Under CDA a and b are the only objects to consider, and the answer is yes. However, if we adopt ODA, and allow for an additional object c, we get the database DB with $a \xrightarrow{R_1} c$ and $c \xrightarrow{R_2} b$, for which $(a, b) \notin ans(R_1 + R_2, DB)$. Hence under ODA the answer is no. ∎

Notwithstanding the difference between the two assumptions, CDA can be reformulated in terms of ODA. It suffices to add an additional view V with $def(V) = \Sigma^*$, $ext(V) = \mathcal{D} \times \mathcal{D}$, and $as(V) = \text{CVA}$. In this way, even under ODA, no additional objects than those already present in $\mathcal{D}$ can be used to construct databases which are consistent with the views, thus realizing CDA. However, as shown in the following sections, the complexity of query answering using views under CDA is in general lower than under ODA. This justifies to consider the two cases separately.

2.4. Possible answers

The problem of query answering using views can be interpreted as checking whether (c, d) is a *certain answer* to Q [2]. On the other hand, we may be interested in checking whether (c, d) is a *possible answer* to Q, i.e., checking whether $(c, d) \in ans(Q, DB)$, for some DB that is consistent with the views.

From the point of view of logic, finding certain answers is a logical implication problem: check whether it logically follows from the information on the views that (c, d) is in the answer set of Q. Similarly, finding possible answers is a consistency problem: check whether assuming that (c, d) is in the answer set of Q does not contradict the information on the views. The following argument illustrates the relationship between the two problems in our framework.

[2]Here we are referring to intersection of relations, and not to intersection of regular languages.

Assump. on dom.	Assump. on views	*Complexity*		
		data	*expression*	*combined*
closed	all sound	*coNP*	*coNP*	*coNP*
	all exact	*coNP*	*coNP*	*coNP*
	arbitrary	*coNP*	*coNP*	*coNP*
open	all sound	*coNP*	*PSPACE*	*PSPACE*
	all exact	*coNP*	*PSPACE*	*PSPACE*
	arbitrary	*coNP*	*PSPACE*	*PSPACE*

Table 1. Summary of complexity results (all bounds are tight)

Suppose we want to check whether (c, d) is a possible answer to the query Q under either CDA or ODA, given the views $V_1, \ldots, V_k$. We consider three new relations R_a, R_b, and R_q, and two new objects a and b. We add to $V_1, \ldots, V_k$ three views V_a, V_b, and V_Q with definitions $def(V_a) = R_a$, $def(V_b) = R_b$, and $def(V_Q) = R_a \cdot Q \cdot R_b$, and extensions $ext(V_a) = \{(a, c)\}$, $ext(V_b) = \{(d, b)\}$, and $ext(V_Q) = \{(a, b)\}$. Each of the new views may be either sound or exact. We ask whether (a, b) is a certain answer to the query R_q. The answer is yes if and only if there is no database which is consistent with V_a, V_b, V_Q and every V_i. Therefore, considering the definitions and the extensions of V_a, V_b, and V_Q, if the answer is yes, then (c, d) is not a possible answer to Q, while if the answer is no, then a database that is consistent with the views exists, and hence (c, d) is a possible answer to Q. This shows that the problem of finding possible answers can be reduced to the one of finding certain answers.

We also remark that checking whether it logically follows from the information on the views that (c, d) is not in the answer set of Q, can be verified by checking whether (c, d) is not a possible answer of Q. Similarly, checking whether the assumption that (c, d) is not in the answer set of Q does not contradict the information on the views, can be verified by checking whether (c, d) is not a certain answer of Q.

In the following, without loss of generality, we consider only the problem of checking whether a pair of objects is a certain answer of a query.

3. Summary of results and related work

The summary of our results on the complexity of answering regular path queries using views is reported in Table 1. Entries with "all sound" (resp., "all exact") in the column named "Assumption on views" refer to the case where all views are assumed to be sound (resp., exact), whereas "arbitrary" means that for each view V, $as(V)$ can be either SVA, CVA, or EVA. Each entry of the table referring to a complexity class C means that the corresponding problem is complete with respect to C.

Our results show that none of the cases can be solved in polynomial time (unless P = NP). This can be explained by observing that, as noted in [6, 2], query answering using views is strictly related to query answering over incomplete databases. Indeed, when we answer the query on the basis of the views, we know only the extensions of the views, and this provides us with only partial information on the database. Moreover, since our query language admits various forms of incomplete information (due to union and transitive closure), there are in general several possible databases that are consistent with the views. The need of considering all such possibilities is a source of complexity for query answering.

Obviously, under CDA, we know at least the set of objects stored in the database, and therefore, our knowledge is more accurate than in the case of ODA. One important feature of CDA is that it is not necessary to conjecture the existence of unknown objects in the database. This provides the intuition of why under CDA the problem is "only" coNP-complete in all cases, for data, expression, and combined complexity.

On the other hand, under ODA, we cannot exclude the possibility that the database contains more objects than those known to be in the views. For combined complexity, this means that we are forced to reason about the definition of the query and the views. Indeed, the problem cannot be less complex than comparing two regular path queries, and this explains the PSPACE lower bound. Interestingly, our algorithms show that the problem does not exceed the PSPACE complexity. Moreover, the data complexity remains in coNP, and therefore, although we are using a query language that is powerful enough to express a (limited) form of recursion, the problem is no more complex than in the case of disjunctions of conjunctive queries [2].

Query answering using views has been extensively investigated in the last years [2, 20, 14, 23, 4]. As we said in the introduction, none of these works provides decidability results for the case where both the query and the views contain recursion.

The work in [2] shares the same goal of this paper. The authors present an analysis of the data complexity of the problem, for the case where the views and the queries are expressed in terms of various languages (conjunctive queries, Datalog, first-order queries, etc.). Note, however, that they do not consider the case of regular path queries. The results presented in [2] show that, for the query languages considered in that paper, EVA complicates the problem. For example, the data complexity of query answering for the case of conjunctive queries is PTIME under SVA and coNP-complete under EVA. This can be explained by noticing that EVA introduces a form of negation, and therefore it may force to reason by cases on the objects stored

in the views. On the contrary, in the case of regular path queries, EVA does not increase the complexity of the problem relative to SVA. In some sense, the expressive power of the query language forces to reason by cases already under SVA, and EVA does not introduce new complexity.

The problem of query answering using views has also been dealt with techniques based on rewriting queries using views [31]: Given a query Q and views $V_1, \ldots, V_k$ with associated definitions $def(V_1), \ldots, def(V_k)$, generate a new query Q' over the alphabet $V_1, \ldots, V_k$ such that for every database DB, first computing the extension $ans(def(V_i), DB)$ of each V_i, and then evaluating Q' on the basis of such extensions, provides the answer to Q over DB. Several papers investigate this problem for the case of conjunctive queries (with or without arithmetic comparisons) [23, 27], queries with aggregates [29, 12, 21], recursive queries [14], queries expressed in Description Logics [6], and queries over semistructured data, both without regular expressions [25], and with regular expressions [10]. Although methods for query rewriting can be adapted to query answering using views [23], the two problems differ in the following sense. Query rewriting has as inputs the view definitions and the query, and aims at re-expressing the query in terms of the views, using a given set of operators. Then, to compute the answer to the original query, the rewritten query is evaluated on the extensions of the views. On the other hand, query answering takes as inputs the view definitions, the view extensions, the view assumptions, and the query, and computes directly the answer to the query.

Note that computing a rewriting is in general costly [23, 10]. However, since such a computation does not depend on the extension of the views, the data complexity of evaluating the rewriting over the view extensions is not influenced by its cost. Section 7 elaborates more on the relationship between the two problems in our framework.

4. Closed domain

We study query answering using views under CDA. We remind the reader that in this case we have complete knowledge on the set of objects stored in the database. This property makes the present case the simplest one in our setting.

Theorem 8 *Answering queries using views under* CDA *is in coNP wrt combined complexity.*

Proof. Let $\mathcal{D}$ be a finite domain, Q be a query, and $V_1, \ldots, V_k$ be views with definitions $def(V_1), \ldots, def(V_k)$, assumptions $as(V_1), \ldots, as(V_k)$, and extensions $ext(V_1), \ldots, ext(V_k)$ such that $\mathcal{D}$ equals the set $\mathcal{D}_V$ of objects in such extensions. To check that a pair (c, d) is not in $ans(Q)$, we guess a database $DB = (\mathcal{D}, \mathcal{E})$ over $\mathcal{D}$ (i.e., we guess the labeled edges of DB), check that DB is consistent with V_i, for $1 \leq i \leq k$, and then check that (c, d) is not in $ans(Q, DB)$. The claim follows from the fact that all checks can be done in polynomial time in the size of $ext(V_1) \cup \cdots \cup ext(V_k)$ and the size of $Q, def(V_1), \ldots, def(V_k)$ by computing the answer set of Q and $V_1, \ldots, V_k$ exploiting the inductive structure of the associated regular expressions. □

We now give a matching lower bound for the problem both wrt expression complexity and wrt data complexity.

Theorem 9 *Answering queries using views under* CDA *is coNP-hard wrt expression complexity.*

The proof of the theorem above is from validity of 3DNF propositional formulae, and uses exact views only. However, the reduction carries through also in the case where all views are assumed to be sound. Hence the coNP lower bound holds also for the special case where all views are sound and the case where all views are exact.

Theorem 10 *Answering queries using views under* CDA *is coNP-hard wrt data complexity.*

The proof of the theorem above is from graph 3-colorability [18] and makes use of a single view under the assumption that it is exact. This shows that the lower bound holds for the special case where all views are exact. However the same argument holds if the same view is assumed to be sound. Hence we obtain also a coNP lower bound for the special case where all views are sound.

Summarizing the results above we can state the following theorem.

Theorem 11 *Answering queries using views under* CDA *is coNP-complete wrt data complexity, expression complexity and combined complexity.*

5. Open domain and sound views

Here we study query answering under ODA in the case where all views are sound. This special case allows for interesting observations concerning the extension of the proposed methods to more general forms of view definitions.

We start by establishing an upper bound for the combined complexity of the problem.

Theorem 12 *Answering queries using sound views under* ODA *is in PSPACE wrt combined complexity.*

Proof. Let Q be a query, $V_1, \ldots, V_k$ be sound views with definitions $def(V_1), \ldots, def(V_k)$ and extensions $ext(V_1), \ldots, ext(V_k)$, and $\mathcal{D}_V$ be the set of objects in such extensions.

Let $A = (\Sigma, S, S_0, \rho, F)$ be a nondeterministic automaton for Q and consider a mapping $h : \mathcal{D}_V \rightarrow 2^S$. We say that h is consistent with A if the following holds: for every view V_i and every pair $(a, b) \in ext(V_i)$ there is a word $w \in L(def(V_i))$ such that $\rho(h(a), w) \subseteq h(b)$[3]. Note that the existence of a word $w \in L(def(V_i))$ such that $\rho(h(a), w) \subseteq h(b)$ can be verified in PSPACE since it amounts to verify whether it is not the case that $L(def(V_i))$ is included in the language accepted by the automaton $A = (\Sigma, S, h(a), \rho, S \setminus h(b))$.

We show that $(c, d) \notin ans(Q)$ if and only if there is a mapping h that is consistent with A such that $S_0 \subseteq h(c)$ and $h(d) \cap F = \emptyset$.

For the "if" direction, given a mapping h satisfying the condition above we construct from h a database DB as follows: for every view V_i and every pair $(a, b) \in ext(V_i)$ we (i) choose a word $w = r_1 \cdots r_n \in L(def(V_i))$ such that $\rho(h(a), w) \subseteq h(b)$ and (ii) introduce in DB a path $a \xrightarrow{r_1} x_1 \cdots x_{n-1} \xrightarrow{r_n} b$, where $x_1, \ldots, x_{n-1}$ are new objects. DB is consistent with the views by construction and it can be verified that $(c, d) \notin ans(Q, DB)$.

For the "only-if" direction, given a database DB that is consistent with the views and such that $(c, d) \notin ans(Q, DB)$ we build a mapping $h' : \mathcal{D} \rightarrow 2^S$ by putting each state in S_0 in $h'(c)$ and repeating the following until h' does not change any more: if there is an edge $x \xrightarrow{r} y$ in DB and $s \in h'(x)$, then add $\rho(s, r)$ to $h'(y)$. Projecting h' on $\mathcal{D}_V$ we obtain a mapping h which is consistent with A and such that $S_0 \subseteq h(c)$ and $h(d) \cap F = \emptyset$.

Finally, since the size of mappings from $\mathcal{D}_V$ to S is $|\mathcal{D}_V| \cdot |S|$, the existence of one satisfying the required conditions can be checked in PSPACE. □

The algorithm used in the proof above is based on constructing a mapping between objects and states of the automaton for the query, which takes into account how the paths in the database that are used to satisfy the views induce transitions on the automaton. The conditions on the mapping are necessary and sufficient for the existence of a counterexample database for $(c, d) \in ans(Q)$.

It is interesting to observe that the algorithm exploits the regularity of the query Q but not the regularity of the view definitions. All that we need is the ability of taking the product of $def(V_i)$ with a finite automaton and then testing for emptiness. Notably, this allows for extending the above algorithm for answering regular path queries using context free views with the same complexity bound.

We now establish a matching lower bound for the expression complexity of the problem.

Theorem 13 *Answering queries using sound views under* ODA *is PSPACE-hard wrt expression complexity.*

Proof. By reduction from regular expression universality, known to be PSPACE-complete [18]. We reduce universality of a regular expressions E to answering query $Q = E$ using a sound view V with $def(V) = \Sigma^*$ and $ext(V) = \{(c, d)\}$. It is easy to verify that $L(\Sigma^*) \subseteq L(E)$ if and only if $(c, d) \in ans(Q)$. □

We now analyze data complexity in the present setting. It turns out that the algorithm in the proof of Theorem 12, which is optimal wrt combined complexity, is also optimal wrt data complexity.

Theorem 14 *Answering queries using sound views under* ODA *is in coNP wrt data complexity.*

Proof. The algorithm in the proof of Theorem 12 provides a coNP upper bound, if the size of the query and of the view definitions is considered to be constant. Indeed, after guessing a mapping $h : \mathcal{D}_V \rightarrow 2^S$, one can check in polynomial time whether h is consistent with A, and whether it satisfies the conditions on the initial and final states of A. □

Theorem 15 *Answering queries using sound views under* ODA *is coNP-hard wrt data complexity.*

Proof. The proof is as the one for Theorem 10, which does not actually rely on CDA and works for sound views. □

6. Open domain and arbitrary views

We now study query answering under ODA in the most general setting where each view may be either sound or exact[4].

Theorem 16 *Answering queries using (arbitrary) views under* ODA *is in PSPACE wrt combined complexity.*

Proof. Let Q be a query, $V_1, \ldots, V_k$ be views with assumptions $as(V_1), \ldots, as(V_k)$, definitions $def(V_1), \ldots, def(V_k)$ and extensions $ext(V_1), \ldots, ext(V_k)$, and $\mathcal{D}_V$ be the set of objects in such extensions.

Let $A_0 = (\Sigma, S_0, s_0, \rho_0, f_0)$ be a nondeterministic automaton for Q (we assume a single starting state and a single accepting state). Let $A_i = (\Sigma, S_i, s_i, \rho_i, f_i)$ be a nondeterministic automaton for $def(V_i)$. Let $T = \bigcup_{0 \leq i \leq k} S_i$, let $T' = T \cup \{0\}$, where $0 \notin T$, and let $S'_i = S_i \cup \{0\}$, for $0 \leq i \leq k$.

Consider a word $w \in \Sigma^*$. Such a word induces a binary relation $H_w \subseteq T' \times T'$ as follows. The pair (s, s') is in H_w if and only if

[3] We have used the extension of ρ to a mapping from $2^S \times \Sigma^*$ to 2^S.

[4] As discussed in Section 2, complete views are reducible to exact views.

- $s, s' \in S_i$ and $s' \in \rho_i(s, w)$;
- $s = 0$, $s' \in S_i$ and $s' \in \rho_i(s_i, w')$ for some nonempty proper suffix w' of w;
- $s' = 0$, $s \in S_i$ and $f_i \in \rho_i(s, w')$ for some nonempty proper prefix w' of w;
- $s = 0$, $s' = 0$ and $f_i \in \rho_i(s_i, w')$ for some nonempty proper suffix w' of a nonempty proper prefix of w.

Intuitively, H_w keeps information about possible runs of each automaton A_i over w, its prefixes and its suffixes. Given a binary relation $H \subseteq T' \times T'$ one can test in PSPACE whether there exists a word w such that $H = H_w$.

Consider a mapping $h : \mathcal{D}_V \times \mathcal{D}_V \rightarrow 2^{T' \times T'}$, i.e., h assigns to every pair $(a, b) \in \mathcal{D}_V \times \mathcal{D}_V$ a binary relation over T'. We say that h is consistent with $A_0, A_1, \ldots, A_k$ if the following holds for each $(a, b) \in \mathcal{D}_V \times \mathcal{D}_V$: if $h(a, b)$ is not empty, then there exists a word $w_{a,b}$ such that $h(a, b) = H_{w_{a,b}}$.

An accepting path for A_i wrt h from a to b is a sequence $a_0, \ldots, a_m$ of elements in $\mathcal{D}_V$, where $a_0 = a$ and $a_m = b$, such that there is a sequence $t_0, \ldots, t_m$ of states in S'_i, where (i) $t_0 = s_i$ or $t_0 = 0$, (ii) $t_m = f_m$ or $t_m = 0$, (iii) $t_1, \ldots, t_{m-1}$ are in S_i, and (iv) $(t_i, t_{i+1}) \in h(a_i, a_{i+1})$, for $0 \leq i < m$. If such an accepting path exists we say that:

- A_i accepts the pair (a, b) if $t_0 = s_i$ and $t_m = f_i$,
- A_i accepts the pair $(0, b)$ if $t_0 = 0$ and $t_m = f_i$,
- A_i accepts the pair $(a, 0)$ if $t_0 = s_i$ and $t_m = 0$,
- A_i accepts the pair $(0, 0)$ if $t_0 = 0$ and $t_m = 0$.

We show that $(c, d) \notin ans(Q)$ if and only if there exists a mapping h such that:

1. h is consistent with $A_0, A_1, \ldots, A_k$;
2. A_0 does not accept the pair (c, d);
3. for each V_i, $1 \leq i \leq k$,
   - if $as(V_i) =$ SVA then A_i accepts a pair (a, b) if $(a, b) \in ext(V_i)$;
   - if $as(V_i) =$ EVA then (i) A_i accepts a pair (a, b) if and only if $(a, b) \in ext(V_i)$ and (ii) A_i does not accept any pair of the form $(0, b)$, $(a, 0)$, or $(0, 0)$.

For the "if" direction, given a mapping h satisfying the conditions above we can obtain from h a minimal mapping h', still satisfying all conditions, by setting $h'(a, b) = \emptyset$ for as many pairs (a, b) as possible. From h' we construct a database DB as follows: for each $(a, b) \in \mathcal{D}_V \times \mathcal{D}_v$ such that $h'(a, b) \neq \emptyset$ we (i) choose a word $w = r_1 \cdots r_n$ such that $h'(a, b) = H_w$ and (ii) introduce in DB a path $a \xrightarrow{r_1} x_1 \cdots x_{n-1} \xrightarrow{r_n} b$, where $x_1, \ldots, x_{n-1}$ are new objects. It can be verified that DB is consistent with the views and that $(c, d) \notin ans(Q, DB)$.

For the "only-if" direction, given a database DB that is consistent with the views and such that $(c, d) \notin ans(Q, DB)$ then there exists a database DB' that is consistent with the views and such that $(c, d) \notin ans(Q, DB)$ of the following form: DB' is composed of a set of paths $a \xrightarrow{r_1} x_1 \cdots x_{n-1} \xrightarrow{r_n} b$, where $a, b \in \mathcal{D}_V$ and $x_1, \ldots, x_{n-1}$ are objects not in $\mathcal{D}_V$ and not occurring on any other path. We define h as follows: for each pair of objects $a, b \in \mathcal{D}_V$,

$$h(a, b) = \begin{cases} H_{r_1 \cdots r_n}, & \text{if } a \xrightarrow{r_1} x_1 \cdots x_{n-1} \xrightarrow{r_n} b \text{ in } DB'. \\ \emptyset, & \text{otherwise} \end{cases}$$

It can be verified that h satisfies the conditions above.

Finally, since the size of mappings from $\mathcal{D}_V \times \mathcal{D}_V$ to $\mathcal{T}' \times T'$ is $|\mathcal{D}_V|^2 \cdot {T'}^2$, the existence of one satisfying the required conditions can be checked in PSPACE. □

The algorithm used in the proof above is based on the same idea of the one used in the proof of Theorem 12 for sound views, namely to construct a mapping between objects and automata states which takes into account how the paths in the database that are used to satisfy the views induce transitions on the automata representing queries. However, while in the case of sound views it was sufficient to track the transitions for the automaton of the query, in the case of exact views one has also to track the transitions for the automata of the views. This is necessary since one has to guarantee that, if a database can be constructed from the mapping, then it actually represents a correct counterexample to $(c, d) \in ans(Q)$, and therefore is consistent with all views. This means that, if a pair (x, y) of objects is not explicitly asserted to be part of the extension of one of the views V_i, then (x, y) is not in the answer set to $def(V_i)$ over the database. The conditions involving pairs of objects (a, b) that appear in the extensions of the views take into account the known objects, while the conditions involving 0 take into account the objects that are not in the extensions of the views but need to be introduced in the database to actually satisfy them.

Obviously, the PSPACE lower bound of answering queries using sound views provides also a lower bound for arbitrary views. Next we show that the same lower bound holds also for the case where all views are exact.

Theorem 17 *Answering queries using exact views under* ODA *is PSPACE-hard wrt expression complexity.*

Proof. By reduction from regular expression universality [18], known to be PSPACE-complete. We reduce universality of a regular expressions E to answering query $Q = R_1 \cdot E \cdot R_2$ using an exact view V with $def(V) = R_1 \cdot \Sigma^* \cdot R_2$ and $ext(V) = \{(c, d)\}$, where Σ is the set of symbols in E, and R_1 and R_2 are new symbols not appearing in Σ. It is easy to verify that $L(\Sigma^*) \subseteq L(E)$ if and only if $(c, d) \in ans(Q)$.

Observe that the proof is a variant of the one for Theorem 13, except that we guarantee that (c, d) is the only pair of objects that satisfy V. □

We now turn to analyzing data complexity in the present setting.

Theorem 18 *Answering queries using (arbitrary) views under* ODA *is in coNP wrt data complexity.*

Proof. The algorithm in the proof of Theorem 16 provides a coNP upper bound, if the size of the query and of the view definitions is considered to be constant. □

Theorem 19 *Answering queries using exact views under* ODA *is coNP-hard wrt data complexity.*

Proof. The proof is the same as that of Theorem 10 which does not actually rely on CDA. □

7. Relationship with query rewriting

As already mentioned in Section 3, query rewriting techniques have been used to solve the query answering problem [23]. Note however, that in the general case query rewriting using views is not sufficient for query answering. In particular, when the rewriting is not exact (i.e., it is not equivalent to the query), it may miss some tuples that are in the answer to a query, as shown by the following example.

Example 20 Let $Q = R_3 \cdot (R_4 + R_5) + R_1 \cdot R_4 + R_2 \cdot R_5$ be a query and let V_1, V_2, V_3 be views such that

$$def(V_1) = R_1 \qquad def(V_2) = R_2 + R_3$$
$$def(V_3) = R_4 + R_5$$

One can verify that the maximal rewriting of Q using $\{V_1, V_2, V_3\}$ is the empty language (which is obviously not exact). However, if $as(V_1) = as(V_2) = as(V_3) =$ EVA and

$$ext(V_1) = \{(c, b)\} \qquad ext(V_2) = \{(c, b)\}$$
$$ext(V_3) = \{(b, d)\}$$

then $(c, d) \in ans(Q)$. ∎

Obviously, an exact rewriting may still miss some tuples of the answer to a query in the case where the views are sound, but not exact.

Example 21 Let $Q = R_1 \cdot R_2$ be a query and let V_1, V_2, V_3 be views such that

$$def(V_1) = R_1 \qquad def(V_2) = R_2$$
$$def(V_3) = R_1 \cdot R_2$$

An exact rewriting of Q wrt $\{V_1, V_2, V_3\}$ is V_3. However, if $as(V_1) = as(V_2) = as(V_3) =$ SVA, and

$$ext(V_1) = \{(c, a)\} \qquad ext(V_2) = \{(a, d)\}$$
$$ext(V_3) = \{(c, b)\}$$

then $(c, d) \in ans(Q)$, but evaluating V_3 on the extensions of the views we do not get this answer[5]. ∎

If the rewriting is exact and the views are exact, we can use such rewriting to solve the query answering problem. Exploiting the results in [10], where rewriting of regular path queries is studied and an algorithm to compute the maximal rewriting and check its exactness is devised, we get the following theorem.

Theorem 22 *Let $V_1, \ldots, V_k$ be exact views, and Q be a query. If there exists an exact rewriting of Q using the views $V_1, \ldots, V_k$, then answering Q using $V_1, \ldots, V_k$ under both* CDA *and* ODA *is NLOGSPACE wrt data complexity and EXPSPACE wrt expression complexity.*

Proof. Let R be an exact rewriting of a query Q in terms of the views $V_1, \ldots, V_k$. We represent R as a finite automaton over the alphabet $V_1, \ldots, V_k$, and we have that $(c, d) \in ans(Q)$ iff there is a sequence $c_0, \ldots, c_m$ of objects and a sequence $s_0, \ldots, s_m$ of states such that

1. $c_0 = c$ and $c_m = d$;
2. s_0 is an initial state and s_m is an accepting state;
3. for $0 \leq i \leq m$, there is a transition labeled by V_{j_i} from s_i to s_{i+1} and the pair $(c_i, c_{i+1}) \in ext(V_{j_i})$.

This can be checked nondeterministically in space logarithmic in the size of the view extensions and the size of the rewriting. Since the size of the rewriting is worst case double exponential in the size of the query [10], the above condition can be verified nondeterministically in space logarithmic in the size of the view extensions and exponential in the size of the query (by constructing the rewriting on the fly [10]). □

Thus, rewriting enables to decrease the data complexity at the expense of expression complexity.

8. Conclusions

We have studied the problem of answering queries using views for the case where both the query and the views are expressed as regular path queries. We have considered different assumptions both on the extensions of the views, and

[5] Evaluating other exact rewritings, namely $V_1 \cdot V_2$ and $V_1 \cdot V_2 + V_3$, on the extensions of the views, would give (c, d) as an answer. However, there is a priory no reason not to choose V_3 as exact rewriting.

on the domain of the database. For each case, we have presented an algorithm and studied its computational complexity. We have proven the lower bound with respect to data, expression, and combined complexity, and we have shown that it matches the complexity of the provided algorithm.

Our results show that the problem is inherently intractable mainly due to the expressive power of the query language. On the other hand, with respect to data complexity, the problem is no more complex than in cases where the query language does not contain recursion, such as nonrecursive Datalog.

In the future, we aim at extending the analysis to other typical features present in full-fledged query languages over semistructured data. In particular, we aim at adding inverse relations in regular path queries, and allowing for conjunctions of regular path queries.

References

[1] S. Abiteboul. Querying semi-structured data. In *Proc. of ICDT'97*, pages 1–18, 1997.

[2] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS'98*, pages 254–265, 1998.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[4] F. N. Afrati, M. Gergatsoulis, and T. Kavalieros. Answering queries using materialized views with disjunction. In *Proc. of ICDT'99*, volume 1540 of *LNCS*, pages 435–452. Springer-Verlag, 1999.

[5] D. Barbará and T. Imieliński. Sleepers and workaholics: Caching strategies in mobile environments. In *Proc. of ACM SIGMOD*, pages 1–12, 1994.

[6] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *Proc. of PODS'97*, pages 99–108, 1997.

[7] P. Buneman. Semistructured data. In *Proc. of PODS'97*, pages 117–121, 1997.

[8] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proc. of ICDT'97*, pages 336–350, 1997.

[9] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization technique for unstructured data. In *Proc. of ACM SIGMOD*, pages 505–516, 1996.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of PODS'99*, pages 194–204, 1999.

[11] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of ICDE'95*, Taipei (Taiwan), 1995.

[12] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of PODS'99*, 1999.

[13] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML. Submission to the World Wide Web Consortium, Aug. 1998. Available at `http://www.w3.org/TR/NOTE-xml-ql`.

[14] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proc. of PODS'97*, pages 109–116, 1997.

[15] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. Catching the boat with strudel: Experiences with a web-site management system. In *Proc. of ACM SIGMOD*, pages 414–425, 1998.

[16] M. F. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proc. of ICDE'98*, pages 14–23, 1998.

[17] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness*. W. H. Freeman and Company, San Francisco (CA, USA), 1979.

[19] M. L. Ginsberg, editor. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann, Los Altos, 1987.

[20] G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of ICDT'99*, volume 1540 of *LNCS*, pages 332–347. Springer-Verlag, 1999.

[21] S. Grumbach, M. Rafanelli, and L. Tininini. Querying aggregate data. In *Proc. of PODS'99*, 1999.

[22] A. Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of VLDB'96*, pages 402–412, 1996.

[23] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. of PODS'95*, pages 95–104, 1995.

[24] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. of ICDT'99*, volume 1540 of *LNCS*, pages 277–295. Springer-Verlag, 1999.

[25] Y. Papakonstantinou and V. Vassalos. Query rewriting using semistructured views. In *Proc. of ACM SIGMOD*, 1999.

[26] D. Quass, A. Rajaraman, I. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proc. of DOOD'95*, pages 319–344. Springer-Verlag, 1995.

[27] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proc. of PODS'95*, 1995.

[28] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., New York, 1978. Republished in [19].

[29] D. Srivastava, S. Dar, H. V. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. of VLDB'96*, pages 318–329, 1996.

[30] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for phyisical data independence. *VLDB Journal*, 5(2):101–118, 1996.

[31] J. D. Ullman. Information integration using logical views. In *Proc. of ICDT'97*, volume 1186 of *LNCS*, pages 19–40. Springer-Verlag, 1997.

[32] M. Y. Vardi. The complexity of relational query languages. In *Proc. of STOC'82*, pages 137–146, 1982.

[33] J. Widom. Special issue on materialized views and data warehousing. *IEEE Bulletin on Data Engineering*, 18(2), 1995.