Query Answering Using Views for Data Integration over the Web

D. Calvanese\textsuperscript{1}, G. De Giacomo\textsuperscript{1}, M. Lenzerini\textsuperscript{1}, M. Y. Vardi\textsuperscript{2}

\textsuperscript{1} Dipartimento di Informatica e Sistemistica
Universit\`a di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it

\textsuperscript{2} Department of Computer Science
Rice University, P.O. Box 1892
Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu

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 a central one in data integration: 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 over the web, data are typically modeled by means of semi-structured mechanisms. Semi-structured data do not fit into rigid, predefined schemas, and are best described by graph-based data models [1]. Methods for extracting information from semi-structured data on the web necessarily incorporate special mechanisms that are not common in traditional database systems. If one looks at the various proposals for query languages over the web [14] and over structured documents [11], one realizes that they all have a common core. Namely, the fundamental querying mechanism that retrieves all pairs of nodes in the graph connected by a path conforming to a regular expression (regular path queries).

In this paper we address the problem of query answering using views in the context of integrating data over the web. According to the above observation, we concentrate on the common core of any query language over the web, and therefore we assume that both the query and the views are expressed in terms of regular paths. In particular, this means that both may contain a (limited) form of recursion. Our goal is to study the computational complexity of the problem, under different assumptions, namely, closed and open domain, and sound, complete, and exact information on view extensions.

Suppose we want to know whether the pair \((c,d)\) is in the answer to the query \(Q\) over the database \(DB\) having only information on the extension of the views \(V_1, \ldots, V_n\). The closed domain assumption states that the database \(DB\) contains exactly the objects stored in the views. In other words, although we do not know the exact form of \(DB\), we know the set of objects stored in it. On the contrary, the open domain assumption leaves the possibility open that \(DB\) contains other objects besides those stored in the views.

Consider now view \(V_i\) and its extension \(\text{ext}(V_i)\). We say that \(V_i\) is sound if \(\text{ext}(V_i)\) is a subset of the objects in \(DB\) that satisfy the definition \(\text{def}(V_i)\) of \(V_i\). In other words, when a view is sound we know a subset of the pairs of objects in \(DB\) 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: if \(V_i\) is complete, then \(\text{ext}(V_i)\) is a superset of the pairs of objects in \(DB\) that satisfy the definition of \(V_i\). Finally, we say that a view is exact if it is both sound and complete.

As pointed out in [15], in data integration, a sound view \(V_i\) corresponds to a data source which is known to produce only (not necessarily all) the answers to the associated query \(\text{def}(V_i)\). On the other hand, a complete view models a source which is known to produce all answers to the associated query, and maybe more. Finally, an exact view 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 have devised the following results:

- We have characterized the lower bound of the problem, under the different assumptions. In particular, it turns out that answering regular path queries using views is coNP-hard with respect to data
complexity. With respect to expression (and hence combined) complexity, the problem is coNP-hard under the closed domain assumption, and PSPACE-hard under the open domain assumption.

- We have shown that all cases are decidable and have provided algorithms whose complexity exactly matches the corresponding lower bound.

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] and in other papers [12] do not report any decidability results when both queries and views contain recursion. So, our results are the first to exhibit decidability in cases where the language for expressing the query and the views allows recursion.

2 Query Answering Using Views

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 semi-structured 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 [6, 5, 1, 21].

Formally, we consider a database as a graph $DB = \langle D, E \rangle$, where the set $D$ of nodes is called the domain of $DB$, and the edges in $E$ are labeled by elements from an alphabet $\Sigma$. We denote an edge from node $x$ to node $y$ labeled by $r$ with $x \buildrel r \over \rightarrow y$.

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.

In this paper we consider regular path queries (which we call simply queries) i.e., queries that denote all the paths corresponding to words of a specified regular language over the alphabet $\Sigma$. Regular path queries are the basic constituents of queries in semi-structured data, and are typically expressed by means of regular expressions [7, 1, 13, 19]. Another possibility to express regular path queries is to use finite automata.

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

Example 3 Refer to the database in Figure 1, and consider the query $(son + dau)^* dau date m$, asking for the pairs $(x, y)$ such that $x$ is a person and $y$ is the month of birth of a female descendant. It is easy to see that $(John, Jun)$ is in the answer set to the query.

We now introduce the problem of query answering using views. Consider a database $DB = \langle D, E \rangle$, and suppose you want to answer a query $Q$ only on the basis of your knowledge about the extension of a set of views $V_1, \ldots, V_n$. The query $Q$ is a regular path query over the alphabet $\Sigma$, and associated to each view $V_i$ we have
• a definition \( \text{def}(V_i) \) in terms of a regular path query over the alphabet \( \Sigma \),
• a set \( \text{ext}(V_i) \) of pairs of elements of \( \mathcal{D} \) which provides the information about the extension of \( V_i \),
• a specification \( \text{as}(V_i) \) of which assumption to adopt for the view \( V_i \), i.e., how to interpret \( \text{ext}(V_i) \) with respect to \( \text{ans}(V_i, DB) \). We describe below the various possibilities that we consider for \( \text{as}(V_i) \).

As pointed out in several papers [2, 15, 17], the above 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. With respect to the knowledge about the objects, we distinguish between:

• **Closed Domain Assumption (CDA):** the exact set of objects in the database \( DB \) is known, and coincides with the set of objects that appear in the views. We say that a database \( \langle \mathcal{D}, \mathcal{E} \rangle \) is consistent with \( \text{ext}(V_1), \ldots, \text{ext}(V_n) \) under CDA if the set of objects appearing in \( \text{ext}(V_1) \cup \cdots \cup \text{ext}(V_n) \) is equal to \( \mathcal{D} \).

• **Open Domain Assumption (ODA):** only a subset of the objects in the database \( DB \) is known. We say that a database \( \langle \mathcal{D}, \mathcal{E} \rangle \) is consistent with \( \text{ext}(V_1), \ldots, \text{ext}(V_n) \) under ODA if the set of objects that appear in \( \text{ext}(V_1) \cup \cdots \cup \text{ext}(V_n) \) is a subset of \( \mathcal{D} \).

With regard to the knowledge about the views, we consider the following three assumptions:

• **Sound View Assumption (SVA):** When a view \( V_i \) satisfies the SVA, written \( \text{as}(V_i) = \text{SVA} \), from the fact that a pair \((a, b)\) is not in \( \text{ext}(V_i) \) one cannot conclude that \((a, b)\) is not in \( \text{ans}(\text{def}(V_i), DB) \). More formally, if \( \text{as}(V_i) = \text{SVA} \), then a database \( DB \) is consistent with \( V_i \) if \( \text{ext}(V_i) \subseteq \text{ans}(\text{def}(V_i), DB) \).

• **Complete View Assumption (CVA):** When a view \( V_i \) satisfies the CVA, written \( \text{as}(V_i) = \text{CVA} \), from the fact that a pair is in \( \text{ext}(V_i) \) one cannot conclude that such a pair is in \( \text{ans}(\text{def}(V_i), DB) \). On the other hand, from the fact that a pair is not in \( \text{ext}(V_i) \) one can conclude that such a pair is not in \( \text{ans}(\text{def}(V_i), DB) \). If \( \text{as}(V_i) = \text{CVA} \), then a database \( DB \) is consistent with \( V_i \) if \( \text{ext}(V_i) \cup \text{ans}(\text{def}(V_i), DB) \).

• **Exact View Assumption (EVA):** For each view \( V_i \) that satisfies the EVA, written \( \text{as}(V_i) = \text{EVA} \), we know that the extension of the view is exactly the pair of objects that satisfy the view. If \( \text{as}(V_i) = \text{EVA} \), then a database \( DB \) is consistent with \( V_i \) if \( \text{ext}(V_i) = \text{ans}(\text{def}(V_i), DB) \).

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

\[
\begin{align*}
\text{def}(V_1) &= \text{son} + \text{dau}, & \text{as}(V_1) &= \text{EVA} \\
\text{def}(V_2) &= (\text{son} + \text{dau})^* \cdot \text{dau} \cdot \text{m}, & \text{as}(V_2) &= \text{SVA} \\
\text{def}(V_3) &= \text{dau}, & \text{as}(V_3) &= \text{CVA}
\end{align*}
\]

If, for example, \( \text{ext}(V_1) \) is the set of nodes connected by \( \text{son} \) or \( \text{dau} \), \( \text{ext}(V_2) = \{\text{John, Jun}\} \), and \( \text{ext}(V_3) = \{\text{Mary, Ann, John, Lea, Tim, Uli, John, Tim}\} \), then the database is consistent with the views.

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

**Definition 5** Let \( \alpha \) be CDA or ODA. The problem of answering queries using views under the domain assumption \( \alpha \) is the following: Given

• \( \text{def}(V_i), \text{ext}(V_i), \) and \( \text{as}(V_i) \), for each \( V_i \) \((1 \leq i \leq n)\),
• a pair of object \( c, d \in \mathcal{D} \) and a query \( Q \),

decide whether \((c, d)\) \( \in \text{ans}(Q) \), i.e., decide whether \((c, d)\) \( \in \text{ans}(Q, DB) \), for each \( DB \) that is consistent with \( \text{ext}(V_1), \ldots, \text{ext}(V_n) \) under \( \alpha \) and that is consistent with every \( V_i \).

The complexity of the problem can be measured in three different ways [25]:

• **Data complexity:** as a function of the size of \( \text{ext}(V_1) \cup \cdots \cup \text{ext}(V_n) \).

• **Expression complexity:** as a function of the size of \( Q \) and of the expressions \( \text{def}(V_1), \ldots, \text{def}(V_n) \).

• **Combined complexity:** as a function of the size of both \( \text{ext}(V_1) \cup \cdots \cup \text{ext}(V_n) \) and the expressions \( Q, \text{def}(V_1), \ldots, \text{def}(V_n) \).
Observe that CDA and ODA are inherently different assumptions. In particular, objects that are not stored in the views may be necessary in proving that a database exists where a pair is not in the answer set of a query. This is illustrated by the following example.

**Example 6** Suppose $\text{def}(V) = R_1; R_2$, $\text{ext}(V) = \{(a, b)\}$, and we want to check whether $(a, b) \in R_1 + R_2$. If we adopt the CDA, then $a$ and $b$ are the only objects to consider, and the answer is yes. However, if we adopt the ODA, and allow for an additional object $c$, we get the following counterexample to the query: $(a, c) \in R_1$ and $(c, b) \in R_2$.

Similarly, the SVA and the EVA are very different assumptions. To see this, it is sufficient to note that if we adopt the 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 7** Consider views $V_1$ and $V_2$ such that $\text{def}(V_1) = R$, $\text{def}(V_2) = R^*$, $\text{ext}(V_1) = \{(a, c)\}$, $\text{ext}(V_2) = \{(a, b)\}$, and $\text{as}(V_2) = \text{EVA}$. Obviously, from the extension of $V_1$ one 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 $(c, d)$ is a certain answer to the query $Q$ under the domain assumption $\alpha$, given the views $V_1, \ldots, V_n$, and suppose that $\text{as}(V_i) = \text{CVA}$. Replace $V_i$ by $V_i'\ast$ such that $\text{def}(V_i') = \text{def}(V_i) + R_{\text{new}}$, $\text{ext}(V_i') = \text{ext}(V_i)$, and $\text{as}(V_i') = \text{EVA}$, where $R_{\text{new}}$ is a new symbol that does not appear in $Q, V_1, \ldots, V_n$. It is easy to see that the new instance of the problem has the same solution as the original one. For this reason we can concentrate our attention on sound and exact views, and ignore the complete view assumption. Note that we cannot apply similar arguments in order to reduce sound views to exact views, because our query language lacks intersection.

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, one may be interested in checking whether $(c, d)$ is a possible answer to $Q$, i.e., checking whether $(c, d) \in \text{ans}(Q, DB)$, for some $DB$ which is consistent with $\text{ext}(V_1), \ldots, \text{ext}(V_n)$ under $\alpha$, and is consistent with every $V_i$.

From the point of view of logic, finding certain answers is a logical implication problem: check whether $(c, d) \in Q$ logically follows from the information on the views. Similarly, finding possible answers is a consistency problem: check whether $(c, d) \in Q$ is consistent with the information on the views. The following argument illustrates the relationship between the two problems in our framework.

Suppose we want to check whether $(c, d)$ is a possible answer to the query $Q$ under the domain assumption $\alpha$, given the views $V_1, \ldots, V_n$. We add to $V_1, \ldots, V_n$ another view $V_Q$ such that $\text{def}(V_Q) = Q$, $\text{ext}(V_Q) = \{(c, d)\}$ and $\text{as}(V_Q) = \text{SVA}$, and we ask whether $(c, d)$ is a certain answer to the query $R_{\text{new}}$, where $R_{\text{new}}$ does not appear in $V_Q, V_1, \ldots, V_n$. If the answer is yes, then the only reason is that there is no $DB$ which is consistent with $\text{ext}(V_Q), \text{ext}(V_1), \ldots, \text{ext}(V_n)$ under $\alpha$, and is consistent with $V_Q$ and every $V_i$, and therefore $(c, d)$ is not a possible answer to $Q$. If the answer is no, then such a $DB$ exists, and $(c, d)$ is obviously a possible answer to $Q$.

The above observation shows that the problem of finding possible answers can be reduced to the one of finding certain answers (provided that we interpret at least one of the views under the SVA). Therefore, we consider only certain answers.

3 Results

We summarize the results we have obtained on the complexity of answering regular path queries using views in Table 1. Entries with "sound" (resp., "exact") in the column labeled "Assumption on views" refer to the case where all views are assumed to be sound (resp., exact), whereas "arbitrary" means that for each view $V$, $\text{as}(V)$ can be arbitrary. Each entry of the table referring to a complexity class $C$ means that the corresponding problem is complete with respect to $C$.

None of the cases can be solved in polynomial time (unless $P=NP$). This can be explained by observing that, as noted in [4, 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

---

1The automata-theoretic techniques used to prove the results can be found in [8]
the views, and this provides us with only partial information on $DB$. 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 coherent with the knowledge about the views. The need of considering all such possibilities is a source of complexity for query answering.

Obviously, under the 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 the CDA is that it is not necessary to conjecture the existence of unknown objects in the database. This provides the intuition of why under the CDA the problem is “only” coNP-complete in all cases, for data, expression, and combined complexity.

On the other hand, under the 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].

### 4 Related Work

Query answering using views has been extensively investigated in the last years [2, 15, 12, 18, 3]. 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, the EVA complicates the problem. For example, the data complexity of query answering for the case of conjunctive queries is PTIME under the SVA and coNP-complete under the EVA. This can be explained by noticing that the 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, the EVA does not increase the complexity of the problem. In some sense, the expressive power of the query language forces to reason by cases already under the SVA, and the EVA does not introduce new complexity.

The problem of query answering using views has also some connection with the problem of rewriting queries using views [24]: Given a query $Q$ and views $V_1, \ldots, V_n$ with associated definitions $\text{def}(V_1), \ldots, \text{def}(V_n)$, generate a new query $Q'$ over the alphabet $V_1, \ldots, V_n$ such that for every database $DB$, first computing the extension $\text{ans}(\text{def}(V_i), DB)$ 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) [18, 22], queries with aggregates [23, 10, 16], recursive queries [12], queries expressed in Description Logics [4], and queries over semistructured data, both without regular expressions [20], and with regular expressions [9]. Although methods for query rewriting can be adapted to the problem of query answering using views [18], the two problems are different. In query rewriting we can only use the view definitions in a way that abstracts from the view extension. On the other hand, in query answering, we not only have the definitions of the views, but also their extensions for a specific database, and we use such view extensions to answer the query over that database.

Note that computing a rewriting is in general costly [18, 9]. 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.

References