

Towards Inconsistency Management in Data Integration Systems

Ariel Fuxman, Renée J. Miller

University of Toronto

{afuxman,miller}@cs.toronto.edu

1 Introduction

Inconsistency management is a fundamental task in any data integration system. Surprisingly, though, the subject has not received much attention in the data integration community. This is probably due to the fact that it is often assumed that the global schema of a system is not given *a priori*, but it is derived as an output of a *schema integration* process [Batini *et al.*, 1986]. Hence, schema integration methodologies are designed to produce global schemas that are consistent with respect to the sources.

However, the result of the schema integration process does not always reflect the semantics that the user has in mind for the global schema. In fact, in many cases, there is no consistent schema that satisfies the requirements of the user. To illustrate this, consider a simple data integration system, in which the Financial and Human Resources departments of a company are integrating their information on employee's salaries. In the company, employees receive only one salary, and this must be reflected in the global database.

Sources

Fin. Dept.: $R_1(\textit{Employee}, \textit{Salary})$
f.d. $\textit{Employee} \rightarrow \textit{Salary}$
HR Dept.: $R_2(\textit{Employee}, \textit{Salary})$
f.d. $\textit{Employee} \rightarrow \textit{Salary}$

Global

$T(\textit{Employee}, \textit{Salary})$
f.d. $\textit{Employee} \rightarrow \textit{Salary}$

Mapping

$R_1(e, s) \subseteq T(e, s)$
 $R_2(e, s) \subseteq T(e, s)$

Both R_1 and R_2 have a functional dependency $\textit{Employee} \rightarrow \textit{Salary}$. Let $\{(John, 1000)\}$ be an instance of R_1 , and $\{(John, 2000), (Mary, 3000)\}$ be an instance of R_2 . Each individual instance satisfies the functional dependency; however the global database $\{(John, 1000), (John, 2000), (Mary, 3000)\}$ does not. In a schema integration methodology, in order to make the global database consistent with respect to the sources, we would need to relax or drop the functional dependency as a constraint of the global schema. However, this contradicts

the company's view of its data: employees receive only one salary. Therefore, the functional dependency must be included in the global schema, rendering the global database inconsistent with respect to the global constraints.

Traditionally, the data integration community deals with this kind of situation by means of cleaning procedures applied to data retrieved from the sources [Bouzeghoub and Lenzerini, 2001]. We argue that, in some cases, this may not be a completely satisfactory solution. For instance, consider the following ways of "cleaning" the source instances:

- Delete the two tuples for *John* from R_1 and R_2 . Then, the global database $\{(Mary, 3000)\}$ would no longer reflect the fact that John is an employee of the company.
- Delete the salaries of *John* from both relations. Then, it would not be possible to know from the global database $\{(John, null), (Mary, 3000)\}$ whether, for instance, there is an employee with the same salary as Mary.
- Delete only one of the tuples for *John*. Since we cannot arbitrarily decide which tuple to delete, we have two options. One option is asking the user what the real salary of John is. In databases where there is a large number of conflicts, the user would be required to answer a massive number of questions; a more automated mechanism would be desirable. Another option, adopted for instance by [Anokhin and Motro, 2001], consists of ranking conflicting tuples based on metadata of the sources (e.g., tuple $(John, 1000)$ comes from a more reliable source than $(John, 2000)$), and defining resolution strategies (e.g., take the maximum, or the average, of the conflicting salaries). However, the user may not have *a priori* knowledge of the potential inconsistencies, and therefore may not be able to provide metadata information or define a resolution strategy that is appropriate for all conflicts.

The above limitations suggest that data cleaning is not always desirable. In some cases, it may not even be possible, since the autonomy of the sources prohibits the required updates. The limitations become even more evident in other application domains, such as peer-to-peer systems. Consider a peer-to-peer setting where peers have agreed to share data. Each peer may need to be able to use the data it receives even if it violates its own internal semantics (that is, the peer's

constraints). In dynamic environments, where peers are autonomous, we may not be able to assume that peers agree on a set of stable shared semantics.

An alternative approach consists of keeping the sources unchanged, but locally resolving inconsistencies at query time. In consequence, it is necessary to define a framework with a clear semantics for consistent answers to queries. In this work we draw upon the semantics of *minimal repairs*, originally defined by Arenas et al. [1999]. Our contributions are the following. First, the problem of retrieving consistent answers under this semantics has been traditionally perceived as very hard (and mostly all the complexity results in the literature are negative [Chomicki and Marcinkowski, 2002; Cali et al., 2003]). In contrast, we identify a class of queries that are indeed tractable, and give an algorithm to answer these queries. Second, we show that, for some queries in this class, the usual approach of rewriting queries is not feasible since there is no first-order rewriting that achieves the desired semantics. This is, to the best of our knowledge, the first inexpressibility result in the area of querying inconsistent databases. These results extend existing work in the context of queries over single inconsistent databases [Arenas et al., 1999; Cali et al., 2003]. Furthermore, we consider the *practical* application of our algorithms, under assumptions that are reasonable within a data integration system.

2 Inconsistency Management Framework

We will first introduce some basic definitions. A *database schema* \mathbf{R} is a finite collection of relation symbols, each of them with an associated arity. A set of *integrity constraints* Σ consists of sentences in a first order language based on the relation symbols of some schema \mathbf{R} . A *database instance* I over \mathbf{R} is a function that associates to each relation symbol R_i of \mathbf{R} a relation $I(R_i)$. A database instance I is *consistent* with respect to a set of integrity constraints Σ if I satisfies Σ in the standard model-theoretic sense, that is $I \models \Sigma$.

We adopt the following definition for a data integration system. A *data integration system* is a quintuple $\langle \mathbf{S}, \Sigma_{\mathbf{S}}, \mathbf{G}, \Sigma_{\mathbf{G}}, \mathcal{M} \rangle$ where \mathbf{S} and \mathbf{G} are the source and global schemas; $\Sigma_{\mathbf{S}}$ and $\Sigma_{\mathbf{G}}$ are their integrity constraints; and \mathcal{M} is a mapping between source and global instances.

We do not assume that the global schema is a virtual view over the sources. Instead, given an instance $I_{\mathbf{S}}$ of \mathbf{S} (consistent with respect to $\Sigma_{\mathbf{S}}$), we assume a materialized instance $I_{\mathbf{G}}$ of \mathbf{G} . This instance can be generated using any *data exchange* technique (e.g., [Fagin et al., 2003]), as long as it respects the mapping \mathcal{M} . Notice that $I_{\mathbf{G}}$ does not necessarily respect the global constraints of $\Sigma_{\mathbf{G}}$. Therefore, given a query on the global schema, we evaluate it on $I_{\mathbf{G}}$, treating it as an inconsistent database with respect to the global constraints. In this sense, we are addressing the problem of consistently querying a single database, as in [Arenas et al., 1999]. Our results are also relevant to settings where the global database is virtual, but in this paper we do not address issues such as query translation into the source schema, or incompleteness of the global instance as a result of local-as-view mappings (in a restricted context, as we explain in Section 4, these have been treated in [Lembo et al., 2002] and [Bertossi et al.,

2002]).

We now explain the semantics that we adopt to consistently evaluate queries on $I_{\mathbf{G}}$. We define the *distance* between two database instances I and I' as their symmetric difference $\Delta(I, I') = (I - I') \cup (I' - I)$. This notion is used in order to define the *repairs* of a database.

Definition 1 (Repair [Arenas et al., 1999]) *Given a set of integrity constraints Σ , and a database instance I we say that a database instance \mathcal{I} is a repair of I wrt Σ iff $\mathcal{I} \models \Sigma$, and $\Delta(I, \mathcal{I})$ is minimal under set inclusion in the class of instances that satisfy Σ , that is, there is no instance I' such that $I' \models \Sigma$ and $\Delta(I, I') \subset \Delta(I, \mathcal{I})$.*

In the example of the introduction, we have $\Sigma_{\mathbf{G}} = \{Employee \rightarrow Salary\}$, $I(R_1) = \{(John, 1000)\}$ and $I(R_2) = \{(John, 2000), (Mary, 3000)\}$. Let $I_{\mathbf{G}} = \{(John, 1000), (John, 2000), (Mary, 3000)\}$, where $I_{\mathbf{G}}$ is inconsistent with respect to $\Sigma_{\mathbf{G}}$. It has two repairs: $\mathcal{I}_1 = \{(John, 1000), (Mary, 3000)\}$ and $\mathcal{I}_2 = \{(John, 2000), (Mary, 3000)\}$. Notice that, for instance, $\{(John, 2000)\}$ and $\{(Mary, 3000)\}$ are not repairs because they do not satisfy the minimality condition.

Definition 2 (Consistent query answer [Arenas et al., 1999])

*Let I be a database instance, possibly not satisfying a set Σ of integrity constraints. Given a query Q on I , we say that a tuple \bar{t} is a consistent answer wrt Σ , denoted $\bar{t} \in \text{consistent}_{\Sigma}(Q, I)$, if for every repair \mathcal{I} of I wrt Σ , $\bar{t} \in Q(\mathcal{I})$. If Q is a closed formula (i.e., a sentence), then *true* is a consistent answer to Q , denoted $\text{consistent}_{\Sigma}(Q, I) = \text{true}$, if for every repair \mathcal{I} of I wrt Σ , $Q(\mathcal{I}) = \text{true}$. We also write $\text{consistent}_{\Sigma}(Q, I) = \text{false}$ to denote that there is some repair \mathcal{I} of I wrt Σ such that $Q(\mathcal{I}) = \text{false}$.*

In other words, a tuple \bar{t} is a consistent answer if it is an answer to the query in all the repairs of the database. Notice that this corresponds to a familiar notion in data integration, *certain answers* [Halevy, 2001], but the set of possible worlds are the repairs, rather than the possible global instances.

Continuing the example, consider the query $Q_1(e) = \exists s : T(e, s)$. The consistent answers to Q_1 on instance I are $\text{consistent}_{\Sigma_{\mathbf{G}}}(Q_1, I) = \{(John), (Mary)\}$. For query $Q_2(e, s) = T(e, s)$, $\text{consistent}_{\Sigma_{\mathbf{G}}}(Q_2, I) = \{(Mary, 3000)\}$. The tuples for *John* do not appear in the answer to Q_2 because they are not present in all the repairs. Intuitively, this reflects the fact that John's salaries are considered to be inconsistent data.

Uncertainty management is a well-studied subject in the areas of information systems and artificial intelligence (for a survey, see [Motro and Smets, 1997]). However, the problem of retrieving consistent answers from inconsistent databases has only recently received some attention. On the one hand, there are a number of approaches based on translating queries into logic programs ([Lembo et al., 2002], [Barcelo and Bertossi, 2003] among others). The most general is the one given in [Barcelo and Bertossi, 2003], which is able to deal with arbitrary first-order queries. However, these approaches are more concerned with finding correct logic programs rather than with producing efficient solutions. In particular, none of the logic programs proposed

run in polynomial time. On the other hand, there is an approach based on *query rewriting* [Arenas *et al.*, 1999]. The idea is, given a query Q and constraints Σ , produce a first-order query Q' such that, for every instance I , $Q'(I)$ returns $\text{consistent}_\Sigma(Q, I)$, the set of consistent answers for Q . The method is clearly tractable in data complexity¹, since the rewritten query Q' is obtained in polynomial time, and in a data-independent way. Furthermore, the method enables the direct reuse of traditional query engines. However, query rewriting has some limitations. First, the method of [Arenas *et al.*, 1999] works only for a restricted class of queries, quantifier-free conjunctive queries. For instance, it works for the query $Q(e, s) = T(e, s)$, but *not* for the query $Q(e) = \exists s : T(e, s)$ that retrieves all employees that have a salary. More importantly, as we shall see, for some tractable conjunctive queries Q , query rewriting is not feasible, since we prove that there is *no* first-order query Q' that can retrieve the consistent answers for Q .

Our work can be seen as a middle ground between the previous approaches. Unlike the logic-programming techniques, we are particularly concerned with the development of efficient algorithms. And, in contrast to the query-rewriting approach, we focus on practical classes of queries that contain some queries that do not admit first-order rewritings.

3 Finding a class of tractable queries

3.1 An intractable query

In order to find algorithms for queries on inconsistent databases, it is necessary to first determine the complexity of the problem. The following is an example of an intractable query. Consider a schema with one relation R , with attributes *Employee*, *Manager* and *Salary*. Let $\Sigma = \{\text{Employee} \rightarrow \text{Manager}, \text{Salary}\}$. Let Q be a query that decides whether there are two employees with the same salary but different manager.

$$Q = \exists e_1, e_2, s, m_1, m_2 : R(e_1, s, m_1) \wedge R(e_2, s, m_2) \wedge m_1 \neq m_2$$

As shown by Chomicki and Marcinkowski [2002], obtaining the consistent answers for Q is a coNP-hard problem. The fact that such a simple query is hard to compute may seem disappointing at first. However, we note that all the negative results in the literature apply only to *specific* queries; thus, they do not rule out the existence of classes of queries that are easier to compute.

In this section, we address the issue of finding a class of natural queries for which the problem is tractable. Clearly, our ultimate goal is to determine the boundary of tractability of the problem. In Section 3.2 we will present and motivate examples of queries that can be computed in polynomial time, and in Section 3.3 we will give some practical considerations for their implementation in a data integration setting. Finally, in Section 3.4 we will generalize the examples to a class of queries.

¹Throughout the paper, we are interested in the *data complexity* of queries, as opposed to their *query complexity*.

3.2 Examples of tractable queries

So far, the only class of queries that are known to be tractable are quantifier-free conjunctive queries [Arenas *et al.*, 1999]. The example of the previous section suggests that the intractability of the queries is due to the existential quantifiers. In this section, we will present examples of queries with existential quantifiers that are tractable. We will concentrate on a schema which, despite its simplicity, allows queries for which it is not trivial to see whether they are tractable.

In what follows, the examples are on a schema with one relation R , and with attributes *Employee* and *Salary*. The constraint that we will consider consists of a key dependency, $\Sigma = \{\text{Employee} \rightarrow \text{Salary}\}$. Consider a query Q_1 that checks whether there are two employees that have the same salary.

$$Q_1 = \exists e_1, e_2, s : R(e_1, s) \wedge R(e_2, s) \wedge e_1 \neq e_2$$

For the instance $I_1 = \{(John, 1000), (John, 2000), (Mary, 1000), (Mary, 2000), (Anna, 1000), (Anna, 3000)\}$, $\text{consistent}_\Sigma(Q_1, I_1) = \text{false}$ because for the repair $\mathcal{I}_1 = \{(John, 1000), (Mary, 2000), (Anna, 3000)\}$, $Q(\mathcal{I}_1) = \text{false}$. On the other hand, for $I_2 = \{(John, 1000), (John, 2000), (Mary, 1000), (Mary, 2000), (Anna, 1000), (Anna, 2000)\}$, $\text{consistent}_\Sigma(Q_1, I_2) = \text{true}$ since for all of the eight repairs \mathcal{I}_2 of I_2 , $Q_1(\mathcal{I}_2) = \text{true}$.

In order to find the consistent answers for Q_1 , we construct a graph for database instance I . In what follows, all the graph-theoretic terminology is taken from [West, 1996].

Definition 3 *Let I be an instance of a schema with one binary relation $R(E, S)$. The graph G of I is a bipartite graph G , with partitions E and S . Partition E and S have one vertex for each value in the active domain of attributes E and S , respectively. The set of edges of G consists of all tuples (e, s) of $I(R)$.*

We use the graph of I to introduce the following necessary and sufficient condition for $\text{consistent}_\Sigma(Q_1, I) = \text{false}$.

Theorem 1 *Let I be an instance of a schema with one binary relation $R(E, S)$, possibly inconsistent wrt a functional dependency $\Sigma = \{E \rightarrow S\}$. Then, $\text{consistent}_\Sigma(Q_1, I) = \text{false}$ iff the graph of I has a perfect matching.*

Proof. \leftarrow Let G be the graph of I . Assume that G has a perfect matching M . We can build an instance \mathcal{I} by creating a tuple in $\mathcal{I}(R)$ for each edge in M . Since M is a matching, each vertex from partition S is incident to at most one edge. Therefore, $Q_1(\mathcal{I}) = \text{false}$. Also, since the matching is perfect, every key appears in $\mathcal{I}(R)$. Consequently, \mathcal{I} is minimal, and therefore it is a repair of I wrt Σ .

\rightarrow Assume that $\text{consistent}_\Sigma(Q_1, I) = \text{false}$. Then, there must exist a repair \mathcal{I} of I wrt Σ such that $Q_1(\mathcal{I}) = \text{false}$. Let G be the graph of I . We can construct a graph G' by selecting the edges of G that correspond to tuples of $\mathcal{I}(R)$. It is easy to see that G' is a perfect matching of G . \square

As an example, we can test the condition on the graphs for I_1 and I_2 . In Figure 1, we show a perfect matching

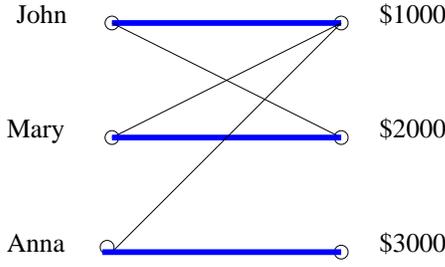


Figure 1: A perfect matching for I_1

for the graph of I_1 . The matching corresponds to repair I_1 . Therefore, $\text{consistent}_\Sigma(Q_1, I_1) = \text{false}$. For I_2 , there is no perfect matching since $|E| > |S|$. Thus, $\text{consistent}_\Sigma(Q_1, I_2) = \text{true}$.

There are a number of algorithms in the literature for deciding the existence of a perfect bipartite matching. For example, one of the best known is given by Hopcroft and Karp [1975], and runs in $O(n^{2.5})$ time. Therefore, Q_1 is a tractable query.

We now show that no approach based on query-rewriting, such as the one of [Arenas *et al.*, 1999], works for Q_1 . This is, to the best of our knowledge, the first inexpressibility result given on queries over inconsistent databases.

Theorem 2 *There is no first-order rewriting Q'_1 of Q_1 such that $\text{consistent}_\Sigma(Q_1, I) = Q'_1(I)$ for every instance I .*

Proof. Let I be an instance of a schema with one binary relation $R(E, S)$. Let G be the graph of I . Assume, towards a contradiction, that there is a first order query Q'_1 such that $\text{consistent}_\Sigma(Q_1, I) = Q'_1(I)$.

Let A_i be the set of vertices of partition S that are neighbors of vertex v_i of partition E in G . A *system of distinct representatives* [Ostrand, 1970] of A_1, \dots, A_n is a sequence of n distinct elements a_1, \dots, a_n with $a_i \in A_i$, $1 \leq i \leq n$. Clearly, G has a perfect matching iff A_1, \dots, A_n has a system of distinct representatives. By Theorem 1, $\text{consistent}_\Sigma(Q_1, I) = \text{false}$ iff G has a perfect matching. Thus, $Q'_1(I) = \text{false}$ iff A_1, \dots, A_n has a system of distinct representatives. But this is a contradiction, since it has been proven in [Libkin and Wong, 1995] that relational algebra, with an appropriate encoding of sets, cannot test whether a family of sets has a system of distinct representatives. \square

Let us continue our examples with query Q_2 , which tests whether there are three employees with the same salary.

$$Q_2 = \exists e_1, e_2, e_3, s : R(e_1, s) \wedge R(e_2, s) \wedge R(e_3, s) \wedge e_1 \neq e_2 \wedge e_2 \neq e_3 \wedge e_1 \neq e_3$$

This query is dealt with in a similar way as Q_1 , but the reduction is now to a *degree-constrained subgraph problem* [Fremuth-Paeger and Jungnickel, 1999], which is a generalization of the perfect matching problem².

²For lack of space, we do not include the proofs of the results in the rest of the paper. All results are proven in the full version [Fuxman and Miller, 2003].

Definition 4 *Let G be a graph with a set of vertices V . Let G' be a subgraph of G on the same set of vertices V , and with a subset of the edges of G . Let $f : V \rightarrow \mathbb{N}$ and $g : V \rightarrow \mathbb{N}$ be two functions such that $f(v) \leq g(v)$, for all $v \in V$. Let $\text{deg}_{G'} : V \rightarrow \mathbb{N}$ be a function such that $\text{deg}_{G'}(v)$ is the degree of v in G' . We say that G' is a (f, g) -degree-constrained subgraph of G if $\text{deg}_{G'}(v) \geq f(v)$ and $\text{deg}_{G'}(v) \leq g(v)$ for all $v \in V$.*

Theorem 3 *Let I be an instance of a schema with one binary relation $R(E, S)$, possibly inconsistent wrt a functional dependency $\Sigma = \{E \rightarrow S\}$. Let G be the graph of I . Let $f(v) = g(v) = 1$ for all vertices of partition E . Let $f(v) = 0$ and $g(v) = 2$ for all vertices of partition S . Then, $\text{consistent}_\Sigma(Q_2, I) = \text{false}$ iff there is a (f, g) -degree-constrained subgraph of G .*

The degree-constrained subgraph problem can be solved by network flow techniques (there are also more specialized and efficient algorithms, see [Fremuth-Paeger and Jungnickel, 1999]). Network flow problems can be solved in polynomial time (e.g., the well-known Ford-Fulkerson algorithm runs in $O(n^3)$ [Ford and Fulkerson, 1958]). Therefore, Q_2 is a tractable query.

3.3 Practical considerations

In the previous examples, the input to our algorithms was the entire database (actually, the graph of the instance). However, in most situations that arise in practice the size of the input will be considerably smaller.

One of the most important reasons is that conflicts are usually confined to a small portion of the database, and it is possible to restrict the input of our algorithms to precisely that portion. To see this, consider a consistent tuple (e, s) (i.e., there is no other tuple (e, s') in the database such that $s \neq s'$). In any perfect matching, s must be associated to e , so we can safely remove the vertices for e and s before computing the perfect matching. For degree-constrained subgraphs, we can remove vertex e from the graph, and reduce the value of $g(s)$ in one unit.

The other factor is inherent to the fact that we are working in a data integration (or data exchange) setting. Recall that in Section 2, in the definition of data integration system, we stated that the source instances I_S are assumed to be consistent with respect to the source constraints Σ_S . Due to the integration process, it is the instance I_G that may become inconsistent with respect to the global constraints Σ_G . Under this assumption, it may be possible to bound the number of conflicts *per key* in I_G by a constant. We can illustrate this with our motivating example. The key *John* has two conflicting tuples: $(John, 1000)$ from relation R_1 , and $(John, 2000)$ from relation R_2 . There cannot be any more conflicts for *John* since there are only two (consistent) relations in the source. Therefore, the number of conflicts per key is bounded by $c = 2$.

This assumption has an important effect on the complexity of the evaluation of queries that have some (though not necessarily all) free variables. Let Q be a query without free variables. Let Q' be query Q , where one of the existential quantifications has been removed. Therefore, Q' has a free

variable e . We can evaluate Q' by instantiating it with each value of free variable e . In this way, we have to evaluate n queries, where n is the cardinality of the active domain of e . However, each of the instantiated queries may be much easier to compute than the original query Q . In fact, we argue that, under the assumption of a constant number of conflicts per key, Q' is easier to compute than Q , even when we take into account all possible instantiations of e .

As an example, consider the query Q_2 of Section 3.2. Let $Q'_2(e_1) = \exists e_2, e_3, s : R(e_1, s) \wedge R(e_2, s) \wedge R(e_3, s) \wedge e_1 \neq e_2 \wedge e_1 \neq e_3 \wedge e_2 \neq e_3$, and the instantiation $Q''_2 = \exists e_2, e_3, s : R(\text{John}, s) \wedge R(e_2, s) \wedge R(e_3, s) \wedge \text{John} \neq e_2 \wedge \text{John} \neq e_3 \wedge e_2 \neq e_3$. When evaluating Q''_2 , the domain of s is no longer the active domain of attribute S , but rather the set of salaries of *John*. Due to our assumption, the cardinality of this set is bounded by some constant c , and it is easy to prove the following proposition.

Proposition 1 *Let I be an instance of a schema with one binary relation $R(E, S)$, possibly inconsistent wrt to $\Sigma = \{E \rightarrow S\}$. Let G be the graph of I . Let $N(\text{John})$ be the set of vertices of partition S that are connected to the vertex for *John* in G . Let c be a constant. Assume $|N(\text{John})| = c$. Let $E' = \{e | e \in E, e \neq \text{John}, \text{ and all edges from } e \text{ are incident into vertices in } N(\text{John})\}$. Let G' be the induced subgraph of G whose set of vertices is $V' = N(\text{John}) \cup E'$. Then, $\text{consistent}_\Sigma(Q''_2, I) = \text{false}$ iff G' has at most $2c$ vertices and G' has a perfect matching.*

The relevance of the above proposition is that Q''_2 can now be evaluated on a graph of constant size, rather than of the size of the database. Since $Q'_2(e)$ must be instantiated for every value of e , it can be evaluated in $O(n)$, as opposed to $O(n^3)$, the running time for Q_2 . A similar reasoning can be applied to all boolean queries. Therefore, queries with free variables are in general easier, never harder, to compute than boolean queries. In fact, in the extreme case when all variables are free, they even admit a first-order rewriting [Arenas *et al.*, 1999].

3.4 A class of tractable queries

We now generalize the previous examples into a class of queries. Intuitively, the existentially-quantified variables of queries in this class can appear in an arbitrary number of predicates, as long as the predicates correspond to the same (binary) relation. Consider the following query Q_3 over the same schema used in the previous section.

$$Q_3 = \exists e_1, e_2, e_3, e_4, e_5, s_1, s_2 : R(e_1, s_1) \wedge R(e_2, s_1) \wedge R(e_3, s_1) \wedge e_1 \neq e_2 \wedge e_1 \neq e_3 \wedge e_2 \neq e_3 \wedge R(e_4, s_2) \wedge R(e_5, s_2) \wedge e_4 \neq e_5 \wedge s_1 \neq s_2$$

The query Q_3 extends Q_2 by considering two non-key elements, s_1 and s_2 , associated to keys of R . We can generalize this with the following class of queries. Consider a schema \mathbf{R} that contains binary relations R_1, R_2, \dots, R_n , each of them possibly inconsistent with respect to their corresponding key dependencies. We have identified the following queries as being tractable:

$$Q = Q_{free} \wedge Q_{exist}^1 \dots \wedge Q_{exist}^n$$

where Q_{free} is a quantifier-free query, and each Q_{exist}^l is a boolean query on relation R_l of the form:

$$Q_{k, j_1, \dots, j_k} = \exists s_1, \dots, s_k : Q_{j_1} \wedge \dots \wedge Q_{j_k} \wedge \text{Distinct}(s_1, \dots, s_k)$$

where

$$Q_{j_i} = \exists e_{i,1}, \dots, e_{i,j_i} : \bigwedge_{1 \leq j \leq j_i} R_l(e_{i,j}, s_i) \wedge \text{Distinct}(e_{i,1}, \dots, e_{i,j_i})$$

The notation $\text{Distinct}(x_1, \dots, x_k)$ is an abbreviation for a formula denoting that x_1, \dots, x_k are all different elements. To see that Q_3 is an instance of the class, let $k = 2$, $j_1 = 3$ and $j_2 = 2$. Also, notice that query Q_1 of Section 3.2 is another instance of the class, where $k = 1$ and $j_1 = 2$. Therefore, no approach based on first-order query-rewriting works for this class of queries.

From the observations of the previous section on free variables, and the results on quantifier-free queries [Arenas *et al.*, 1999], it follows directly that free variables do not make the problem intractable. Also, since the existentially-quantified subqueries do not share variables, they can be treated independently of each other. Therefore, for the sake of clarity, we will now concentrate on a boolean query Q_{exist}^l . The algorithm can be easily extended for the evaluation of any query in the class. The relevance of the algorithm is that it illustrates a tractable approach to consistent query answering. However, by no means do we suggest this the most efficient algorithm that can be obtained.

Input: Query Q_{k, j_1, \dots, j_k}
 Bipartite graph G with partitions E and S
Output: true/false
for $m = 1$ to k **do**
 for each set K of $m-1$ distinct values of S **do**
 Let $f(v)=1$ for all $v \in E$;
 $f(v)=0$ for all $v \in S$
 Let $g(v)=1$ for all $v \in E$;
 $g(v)=\infty$ for all $v \in K$;
 $g(v)=j_m - 1$ for all $v \in S - K$
 if there is a $(f-g)$ -degree constrained subgraph of G
 then
 Return false
 end if
end for
end for
 Return true

The total number of degree-constrained subgraph problems that must be solved in the worst case is $O(n^{k-1})$. The value k is a constant, because it depends on the query, not the data, and it can be assumed to be small. Assuming that the degree-constrained subgraph problems are solved with network flow techniques (which run in $O(n^3)$), the total running time of the algorithm is $O(n^{k+2})$. Therefore, all the queries in the class are tractable. The correctness of the algorithm is proven in the full version of this paper [Fuxman and Miller, 2003].

4 Final Remarks

We are working on larger classes of queries than the ones presented here. In particular, at the moment we are concentrating on a class in which existentially-quantified variables can occur in the literals corresponding to not just one, but an arbitrary number of binary relations, each of them with a functional dependency. Once we settle the issue on binary relations, we will start working on queries with existentially-quantified variables in relations of arbitrary arity. Given the results in [Chomicki and Marcinkowski, 2002], some of these classes are not tractable. For them, we are exploring the application of approximation algorithms.

As mentioned in Section 2, for the moment we are not considering the global schema as a virtual view of the system. To the best of our knowledge, the only references in the literature that address this issue in conjunction with consistent query answering are Bertossi et al. [2002] and Lembo et al. [2002]. The approach of Bertossi et al. consists of rewriting a query on the global schema into another first-order query by using the technique given in [Arenas et al., 1999]. Since the rewritten queries may contain negation, the authors provide rules for producing query plans that can cope with negation. The limitation of this approach is that we have shown that some simple queries do not admit first-order rewritings. Lembo et al. adopt a logic programming approach, which consists of conjoining a program that specifies repairs of the sources with a program that computes the query. However, when the actual logic program gets executed, in the worst case it may need to materialize a set of repairs exponentially larger than the size of the sources. In contrast, our approach is specifically designed to avoid materializing all repairs.

Finally, we will also consider the case of global schemas with inclusion dependencies, in addition to functional dependencies. Notice that, while the repairs for functional dependencies only delete tuples from the original instance, the repairs for an inclusion dependency may either insert or delete tuples. This corresponds to either an *open* or *closed* interpretation of the sources. Although most research on data integration concentrates on the open interpretation, both interpretations become relevant when dealing with inconsistencies. For some results on inclusion dependencies on inconsistent databases, see [Cali et al., 2003].

Acknowledgments. We would like to thank Leonid Libkin, Leopoldo Bertossi, Derek Corneil, and Alberto Mendelzon for the insightful discussions and feedback.

References

[Anokhin and Motro, 2001] P. Anokhin and A. Motro. Data integration: Inconsistency detection and resolution based on source properties. In *Workshop on Foundations of Models for Information Integration (FMII)*, 2001.

[Arenas et al., 1999] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. ACM PODS*, pages 68–79, 1999.

[Barcelo and Bertossi, 2003] P. Barcelo and L. Bertossi. Logic programs for querying inconsistent databases. In *Proc. PADL*, Springer LNCS 2562, pages 208–222, 2003.

[Batini et al., 1986] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[Bertossi et al., 2002] L. Bertossi, J. Chomicki, A. Cortes, and C. Gutierrez. Consistent answers from integrated data sources. In *5th Intl. Conf. on Flexible Query Answering Systems*, pages 71–85. Springer LNAI 2522, 2002.

[Bouzeghoub and Lenzerini, 2001] M. Bouzeghoub and M. Lenzerini. Introduction to the special issue on data extraction, cleaning and reconciliation. *Information Systems*, 26(8):535–536, 2001.

[Cali et al., 2003] A. Cali, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. To appear in *PODS*, 2003.

[Chomicki and Marcinkowski, 2002] J. Chomicki and J. Marcinkowski. On the computational complexity of consistent query answers. *coRR cs.DB/0204010*, 2002.

[Fagin et al., 2003] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

[Ford and Fulkerson, 1958] L. Ford and D. Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 10:78–85, 1958.

[Fremuth-Paeger and Jungnickel, 1999] C. Fremuth-Paeger and D. Jungnickel. Balanced network flows. I. A unifying framework for design and analysis of matching algorithms. *Networks*, 33(1):1–28, 1999.

[Fuxman and Miller, 2003] A. Fuxman and R. J. Miller. Towards inconsistency management in data integration systems. Technical Report CSRG-470, University of Toronto, 2003.

[Halevy, 2001] A. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[Hopcroft and Karp, 1975] J. Hopcroft and R. M. Karp. An $O(n^{2.5})$ algorithm for maximum matching in bipartite graphs. *SIAM J. of Computing*, 2:225–231, 1975.

[Lembo et al., 2002] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *KRDB*, 2002.

[Libkin and Wong, 1995] Leonid Libkin and Limsoon Wong. On representation and querying incomplete information in databases with bags. *Information Processing Letters*, 56(4):209–214, 1995.

[Motro and Smets, 1997] A. Motro and P. Smets, editors. *Uncertainty Management in Information Systems. From Needs to Solutions*. Kluwer Academic Publishers, 1997.

[Ostrand, 1970] P. Ostrand. Systems of distinct representatives. *J. of Math. Analysis and Applications*, 32:1–4, 1970.

[West, 1996] D. West. *Introduction to Graph Theory*. Prentice Hall, 1996.