Efficient Application of Answer Set Programming for Advanced Data Integration*

Nicola Leone, Francesco Ricca, Luca Agostino Rubino, and Giorgio Terracina

Dipartimento di Matematica, Università della Calabria, 87030 Rende, Italy
{leone,ricca,terracina,rubino}@mat.unical.it

Abstract. An information integration system combines data residing at different sources, providing the user with a unified view of them, called *global schema*. When some constraints are imposed on the quality of the global data, the integration process becomes difficult and, in some cases, it may be unable to provide consistent results to user queries. The database community has spent many efforts in this area, relevant research results have been obtained to clarify semantics, decidability, and complexity of data-integration under constraints (often called consistent query answering - CQA). However, while efficient systems are already available for simple data integration scenarios, scalable solutions have not been implemented yet for advanced data-integration under constraints. This paper provides a contribution in this setting: it starts from state of the art techniques to carry out consistent query answering and proposes optimized solutions; these have been implemented in a efficient system based on Answer Set Programming (a purely declarative logic programming formalism). Experimental activities conducted in a real world scenario and reported in the paper confirm the effectiveness of the approach.

1 Introduction

The task of an *information integration system* is to combine data residing at different sources, providing the user with a unified view of them, called *global schema*. Users can formulate queries in a transparent and declarative way over the global schema, they do not need to know any information about the sources. The information integration system automatically retrieves the relevant data from the sources, and suitably combines them to provide answers to user queries.

Recent developments in IT, such as the expansion of the Internet, have made available to users a huge number of information sources, generally autonomous, heterogeneous and widely distributed. As a consequence, information integration has emerged as a crucial issue in several application domains, e.g., distributed databases, cooperative information systems, data warehousing, or on-demand computing.

In many cases the application domain requires to impose some constraints on the integrated data. For instance, it may be at least desirable to impose some keys on global relations (i.e., on the relations of the global scheme).

* This work has been partially supported by the Calabrian Region under PIA (Pacchetti Integrati di Agevolazione industria, artigianato e servizi) project DLVSYSTEM approved in BURC n. 20 parte III del 15/05/2009 - DR n. 7373 del 06/05/2009.

As an example, suppose one needs to merge the lists of students from two different universities, the set of their IDs may overlap since they have been assigned independently by the two universities. This may cause ID duplications in the global database after that the merging process has been carried out. If the student ID must be a key in the global database, then some corrective actions must be carried out in order to avoid the generation of an inconsistent global database (i.e., in the database over the global scheme, resulting from the integration). Such corrective actions are usually called *database repairs* in the literature [1–4]. An information integration system should be able to return all and only the consistent answers, that is the answers which are true in every repair of the database (this is called *Consistent Query Answering - CQA*) [1]. The bad news is that, in most cases, several repairs are possible for each violation of a constraint, making information integration a computationally difficult task: consistent query answering is co-NP-hard even in very simple settings, like the example above, where only a single key constraint is present on the global scheme. Moreover, it has been shown that mixing different kinds of constraints (e.g. denial constraints, inclusion and exclusion dependencies) on the same global database may easily make the query answering process undecidable [5].

The database community has spent many efforts in this area, relevant research results have been obtained to clarify semantics, decidability, and complexity of data-integration under constraints.

However, while efficient systems are already available for simple data integration scenarios, scalable solutions have not been implemented yet for advanced data-integration under constraints, mainly due to the fact that handling inconsistencies arising from constraints violations is inherently hard.

This paper provides a contribution in this setting. Specifically, it starts from practical applications of state-of-the-art approaches to provide well-tuned optimizations techniques aiming at "localizing" and limiting the inefficient computation, due to the handling of inconsistencies, to a very small fragment of the input, yet allowing interesting classes of constraints.

The presented work takes advantage from the experience we gained in the INFOMIX [6] project, and overcomes some limitations we experienced in real-world scenarios. In fact, our main goal is to provide a purely declarative, logic-based solution to the problem of data integration under constraint, which is efficient and can be profitably used also in real-world applications.

The main characteristics of the proposed approach are the following:

- It supports a powerful and comprehensive information integration model, which is based on a formal and purely declarative semantics. The knowledge about the integration domain can be easily specified. In particular, it allows: (i) the possibility of defining expressive integrity constraints (ICs) over the global schema, (ii) the precise characterization of the relationship between global schema and the local data sources, (iii) the formal definition of the underlying semantics, (iv) as well as the use of a powerful query language.
- It is based on Answer Set Programming (ASP) and exploits datalog-based methods for answering user queries, which are sound and complete with respect to the semantic of query answering. The problem of consistent query answering is reduced

to cautious reasoning on disjunctive datalog programs; this allows to effectively compute the query results precisely, by using a state-of-the-art disjunctive datalog system. The formal query semantics is captured also in presence of inconsistent data.
- It allows to obtain fast query-answering, even in such a powerful data-integration framework, thanks to the novel combination of a number of optimization techniques that tend to minimize the inefficient computation.
- In order to handle large amounts of data, usually involved in real-world integration scenarios, it adopts as internal query evaluation engine the disjunctive datalog system DLV^{DB} [7, 8] which allows for mass-memory evaluations and distributed data management features.

In order to asses the effectiveness of the proposed optimizations, we carried out a thorough experimental activity on a real world scenario. Obtained results, reported in the paper, are encouraging and confirm our intuitions.

The plan of the paper is as follows. Section 2 formally introduce the data integration model and the consistent query answering problem considered in the paper. Section 3 first introduces a standard approach to handle CQA with ASP and then presents some optimizations. Section 4 outlines some of the features of the system we developed on the proposed approach whereas Section 5 introduces the benchmark framework we adopted in the tests and presents obtained results. Finally, in Section 6 we draw some conclusions.

2 The Data Integration Context

2.1 The Data Integration Model

In our setting, a data integration system [1] $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is the global schema, which provides a uniform view of the information sources to be integrated, $\mathcal{S}$ is the source schema, which comprises the schemas of all the sources to be integrated, and $\mathcal{M}$ is the mapping establishing a relationship between $\mathcal{G}$ and $\mathcal{S}$. $\mathcal{G}$ may contain integrity constraints (ICs). $\mathcal{M}$ is a *Global-As-View* (GAV) mapping [1], i.e., $\mathcal{M}$ is a set of logical implications $\forall x_1 \cdots \forall x_n . \Phi_{\mathcal{S}}(x_1, \ldots, x_n) \supset g_n(x_1, \ldots, x_n)$, where g_n is a relation from $\mathcal{G}$, n is the arity of g_n, $\Phi_{\mathcal{S}}$ is a conjunction of atoms on $\mathcal{S}$ and $x_1, \ldots, x_n$ are the free variables of $\Phi_{\mathcal{S}}$. Each global relation is thus associated with a *union of conjunctive queries* (UCQs). Both $\mathcal{G}$ and $\mathcal{S}$ are assumed to be represented in the relational model, whereas $\mathcal{M}$ is represented as a set of datalog rules.

As an example consider a bank association that desires to unify the databases of two branches. The first database models managers by using a table $man(code, name)$ and employees by a table $emp(code, name)$, where code is a primary key for both tables. The second database stores the same data in table $employees(code, name, role)$. Suppose that the data has to be integrated in a global schema with two tables: $m(code)$, and $e(code, name)$, having both code and name as keys and the inclusion dependency

$m[code] \subseteq e[code]$, indicating that manager codes must be employee codes. GAV mappings are defined as follows:[1]

$$e(C, N) :- emp(C, N). \quad e(C, N) :- employee(C, N, \_).$$
$$m(C) :- man(C, \_). \qquad m(C) :- employee(C, \_, \text{`}manager\text{'}).$$

If emp stores $(e1, john)$, $(e2, mary)$, $(e3, willy)$, man stores $(e1, john)$, and $employees$ stores $(e1, ann, man)$, $(e2, mary, man)$, $(e3, rose, emp)$, it is easy to verify that, while the source databases are consistent w.r.t. local constraints, the global database obtained by evaluating the mappings violates the key constraint on e (e.g. both $john$ and ann have the same code $e1$ in table e). Basically, when data are combined in a unified schema with its own integrity constraints the resulting global database might be inconsistent; any query posed on an inconsistent database would then produce an empty result.

In this context, user queries must be re-modelled according to the mappings and violated constraints, in order to compute *consistent* answers, i.e. answers which consider as much as possible of *correct* input data.

2.2 Consistent Query Answering

In the field of data-integration several notions of consistent query answering have been proposed (see [3] for a survey), depending on whether the information in the database is assumed to be *correct* or *complete*. Basically, the incompleteness assumption coincides with the *open world assumption*, where facts missing from the database are not assumed to be false. In our approach, we assume that sources are complete; as argued in [4], this choice strengthens the notion of minimal distance from the original information.[2] Moreover, there are two important consequences of this choice: database repairs can be obtained by only *deleting tuples* and, thus, computing CQA for conjunctive queries remains *decidable* even for arbitrary sets of denial constraints and inclusion dependencies [4] which are the most common schema constraints.

More formally, given a global schema $\mathcal{G}$ and a set C of integrity constraints, let $\mathcal{DB}$ and $\mathcal{DB}^r$ be two global database instances. $\mathcal{DB}^r$ is a *repair* [4] of $\mathcal{DB}$ w.r.t. C, if $\mathcal{DB}^r$ satisfies all the constraints in C and the instances in $\mathcal{DB}^r$ are a maximal subset of the instances in $\mathcal{DB}$. Basically, given a conjunctive query Q, *consistent answers* are those query results that are not affected by constraint violations and are true in any possible repair [4]. Thus, given a database instance $\mathcal{DB}$ and a set of constraints C, a conjunctive query Q is consistently true in $\mathcal{DB}$ w.r.t. C if Q is true in every repair of $\mathcal{DB}$ w.r.t. C. Moreover, if Q is non-ground, the consistent answers to Q are all the tuples $\overline{t}$ such that the ground query $Q[\overline{t}]$ obtained by replacing the variables of Q by constants in $\overline{t}$ is consistently true in $\mathcal{DB}$ w.r.t. C.

Following the example introduced in the previous Section, the global database has the following four repairs:

[1] In the examples we denote mappings by datalog rules. For instance $e(C, N) :- emp(C, N).$ stands for $\forall C \forall N e(C, N) \supset emp(C, N)$.

[2] It is worth noting that, in relevant cases like denial constraints, query results coincide for both correct and complete information assumptions.

$\mathcal{DB}_1^r = \{e(e2, mary), e(e1, john), e(e3, willly), m(e1), m(e2)\}$
$\mathcal{DB}_2^r = \{e(e2, mary), e(e1, john), e(e3, rose), m(e1), m(e2)\}$
$\mathcal{DB}_3^r = \{e(e2, mary), e(e1, ann), e(e3, willly), m(e1), m(e2)\}$
$\mathcal{DB}_4^r = \{e(e2, mary), e(e1, ann), e(e3, rose), m(e1), m(e2)\}$

Moreover, the query $Q = m(X)?$, asking for the list of manager codes, has both $e1$ and $e2$ as consistent answers.

In the next section, we show how Answer Set Programming (ASP) can be exploited for efficiently computing consistent answers to user queries. We assume that the reader is familiar with ASP.

3 Consistent Query Answering via ASP

Answer Set Programming [9, 10] is a powerful logic programming paradigm allowing (in its general form) for disjunction in rule heads [11] and nonmonotonic negation in rule bodies. ASP is a purely declarative language that can represent every problem in the complexity class Σ_2^P and Π_2^P (under brave and cautious reasoning, respectively [12]).

The suitability of ASP for implementing CQA has been already recognized in the literature [3, 6]. The idea is to produce an ASP program Π_{cqa} having an answer set for each repair, so that the problem of computing CQA corresponds to cautious reasoning on Π_{cqa}. Formally, given a global database $\mathcal{DB}$, a set of integrity constraints C and a conjunctive query Q,[3] we produce an ASP program Π_{cqa} and a query Q_{cqa}, such that: Q is consistently true in $\mathcal{DB}$ w.r.t. C iff Q_{cqa} is true in every answer set of Π_{cqa}, in symbols: $\Pi_{cqa} \models_c Q_{cqa}$. In other words, Q is consistently true iff Q_{cqa} is a *cautious consequence* of Π_{cqa}.

In our setting, the most common schema constraints can be expressed in ASP as follows:

(c_1) $\quad :- a_1(t_1), \cdots, a_n(t_n), \sigma(t_1, \ldots, t_n).$
(c_2) $\quad :- a_1(t), not\ \ aux_{a_2(t)}(t). \qquad aux_{a_2(t)}(t) :- a_2(t, t').$

where t_i is a tuple and $\sigma(t_1, \ldots, t_n)$ is a conjunction of comparison literals of the form $X\theta Y$ with $\theta \in \{<, >, =, \neq\}$, and $aux_{a_2(t)}(t)$ is a fresh new auxiliary predicate defining a projection on a_2. Constraints of type c_1 are called denial constraints; whereas constraints of type c_2 model inclusion dependencies under the assumption of complete sources. In particular, we allow only acyclic[4] inclusion dependencies, which are the most common ones, to limit the complexity of CQA to co-NP, see [4]. Moreover, note that key constraints are special cases of denial ones.

For instance, in the example of Section 2, we considered the following three global constraints:

[3] As usual, a conjunctive query of arity n is a closed formula the form $q(x_1, \ldots, x_n) :- conj(x_1, \ldots, x_n, y_1, \ldots, y_k)$. where $conj$ is a conjunction of atoms involving variables $x_1, \ldots, x_n, y_1, \ldots, y_k$; sometimes if $k = 0$ we write only $conj(x_1, \ldots, x_n)$?

[4] Informally, a set of inclusion dependencies is acyclic if no attribute of a relation R transitively depends (w.r.t. inclusion dependencies) on an attribute of the same R.

$:- e(X,Y), e(X,Z), Y \neq Z.$ $\quad :- e(X,Y), e(Z,Y), X \neq Z.$
$:- m(X), not\ \ code(X).$ $\quad code(X) :- e(X,Y).$

respectively requiring that both code and name are keys for *e* and that $m[c] \subseteq e[c]$; $code$ is an auxiliary predicate computing the projection of e on its first attribute.

In the following we introduce two algorithms that take as input a data integration system and a query and produce an ASP program that can be exploited for computing CQA. First we describe a standard algorithm producing a general encoding of a CQA problem in ASP; then we propose a new "optimized" method that is able to produce programs complexity-wise optimal according to the complexity classification of constraints and queries of [4].

Standard Solution. Given a global schema having a set of constraints C and a query Q, a general algorithm for building the program Π_{cqa} and the query Q_{cqa} is composed by the following steps:

1- for each constraint of the form c_1 in C, insert the following rule into Π_{cqa}:
$\overline{a}_1(t_1) \vee \cdots \vee \overline{a}_n(t_n) :- a_1(t_1), \cdots, a_n(t_n), \sigma(t_1, \ldots, t_n).$
2- for each atom $a(t)$ occurring in some constraint of C, insert into Π_{cqa} a rule:
$a^*(t) :- a(t), not\ \ \overline{a}(t).$
3- for all constraints of the form c_2 in C, insert the following rules in Π_{cqa}:
$\overline{\overline{a_1}}(t) :- a_1^*(t), not\ \ aux^r_{a_2(t)}(t). \quad aux^r_{a_2(t)}(t) :- a_2^r(t, t').$
4- for each $a(t)$ occurring in some constraint of C insert into Π_{cqa} the following rule:
$a^r(t) :- a^*(t), not\ \ \overline{a}(t), not\ \ \overline{\overline{a}}(t).$
5- build Q_{cqa} form Q by replacing each $a(t)$ by $a^r(t)$ whenever $a(t)$ occurs in some constraint in C.

Intuitively, the disjunctive rules (step 1) guess the tuples to be deleted (step 2) for satisfying denial constraints. Rules generated by step 3, remove tuples violating also referential integrity constraints; eventually, step 4 builds repaired relations. Note that the minimality of answer sets guarantees that deletions are minimized.

In our ongoing example, the program obtained by applying the algorithm above is:

$\overline{e}(X,Y) \vee \overline{e}(X,Z) :- e(X,Y), e(X,Z), Y \neq Z.$
$\overline{e}(X,Y) \vee \overline{e}(Z,Y) :- e(X,Y), e(Z,Y), X \neq Z.$
$e^*(t) :- e(t), not\ \ \overline{e}(t). \quad m^*(X) :- m(X), not\ \ \overline{m}(X).$
$code^r$ **(X)** $:- e^r$ **(X,Y).**
$\overline{\overline{m}}(X) :- m^*(X), not\ \ code^r(X).$
$e^r(X,Y) :- e(X,Y), not\ \ \overline{e}(X,Y), not\ \ \overline{\overline{e}}(X,Y).$
$m^r(X) :- m(X), not\ \ \overline{m}(X), not\ \ \overline{\overline{m}}(X).$
$m^r(X)?$

When this program is evaluated on the database facts we obtain four answer sets. It can be verified that, all the answer sets contain $m^r(e1)$ and $m^r(e2)$, (i.e., they are cautious consequences of Π_{cqa}) and, thus, $m(e1)$ and $m(e2)$ are consistent answers to the original query.

It can be shown that this algorithm always finds a repair for the database (and thus is able to compute query answer) which can possibly be, in the worst case, empty.

Optimized Solution. The algorithm reported above is a general solution for solving the CQA problem, but, in several cases, more efficient ASP programs can be produced. First of all note that the general algorithm blindly considers all the constraints on the global schema, including those that have no effect on the specific query. Consequently, redundant logic rules might be produced which slow down program evaluation. Note also that, there are a number of cases in which, according to [4], the complexity of CQA stays in P; but disjunctive programs, for which cautious reasoning is an hard task [12], are generated in presence of denial constraints. This means that, the evaluation of the produced logic programs might be much more expensive than required in those "easy" cases. In more detail, depending on the types of both schema constraints and queries, CQA is tractable in the following cases:

- *Quantifier-free queries* and either:
  - denial constraints only, or
  - at most one key per relation;
- *Simple Conjunctive queries* and either:
  - at most one functional dependency per relation, or
  - at most one key per relation
- *Conjunctive queries* and:
  - inclusion dependencies only

where *quantifier free queries* are those that do not contain projections operations, *simple conjunctive queries* are those without repeated relation symbols and with limited variable sharing (joins are not admitted).

In the following we provide an optimized version of the standard algorithm that is capable of identifying tractable (sub-)cases for a generic input query and that produces ASP programs for CQA which are non-redundant and complexity-wise optimal.

Given a global schema $\mathcal{G}$, a set of constraints C on $\mathcal{G}$ and a query Q, the optimized algorithm analyzes both C and Q and: (i) singles out only the constraints affecting query results, and (ii) employs positive non-disjunctive rules for dealing with denial constraints in known tractable cases.

Specifically, a directed labelled graph $G_c = \langle N, E \rangle$, called *constraint graph*, is first built. G_c contains a node $n \in N$ for each relation in $\mathcal{G}$, and an arc $e = (p, q, c)$ for each pair of relations $\langle p, q \rangle$ involved in a global constraint $c \in C$. In more detail, G_c is built from $\mathcal{G}$ and C as follows: for each $c \in C$: if c is a denial constraint of the form $:- p_1(t_1), \cdots, p_k(t_k), \sigma(t_1, \ldots, t_k)$ an arc (p_i, p_j, c) is added to E for each $i, j \in [1, k]$ with $i \neq j$; whereas, if c is an inclusion dependency of the form $:- p(t), not\ \ q(t)$ then an arc (p, q, c) is added to E.

After analyzing and classifying the query (to recognize whether it is either quantifier-free, or simple conjunctive, or conjunctive), the constraint graph G_c is visited several times starting from each relation in the query. The visited nodes of G_c correspond to the relations involved in the query process, whereas the arcs traversed during the visits correspond to the constraints that might influence the query results. Thus, the corresponding relations and constraints are marked to be considered for further processing; unmarked constraints will be discarded. At the same time, the algorithm tags each marked constraint to be either easy or hard, depending on whether the above-reported

conditions on the complexity of CQA are satisfied or not. In particular, the tag associated to a given constraint is set (or updated) during each visit depending on query kind, number and type of encountered constraints. The tag of each constraint c corresponding to a traversed arc e is set to "easy" if both (i) c was not previously tagged as "hard", and (ii) at least one of the following conditions holds (otherwise c is tagged as "hard"):

1. if the query is quantifier-free, and either
   a. all the arcs belonging to the connected component of G_c containing e are labeled by denial constraints, or
   b. all the nodes belonging to the connected component of G_c containing e have at most one outgoing arc labelled by a key constraint
2. if the query is simple-conjunctive, and either
   a. all the nodes belonging to the connected component of G_c containing e have at most one outgoing arc labelled by a functional dependency constraint, or
   b. all the nodes belonging to the connected component of G_c containing e have at most one outgoing arc labelled by a key constraint
3. if the query is conjunctive, and:
   a. all the arcs belonging to the connected component of G_c containing e are labeled by inclusion dependencies

At this point, the ASP program Π_{cqa} is generated as follows:

1- for each denial constraint of the form c_1 which is marked as "hard", insert the following rule into Π_{cqa}:
$\overline{a}_1(t_1) \vee \cdots \vee \overline{a}_n(t_n) :- a_1(t_1), \cdots, a_n(t_n), \sigma(t_1, \ldots, t_n)$.
2- for each denial constraint of the form c_1 which is marked "easy", insert the following n rules into Π_{cqa}:
$\overline{a}_1(t_1) :- a_1(t_1), \cdots, a_n(t_n), \sigma(t_1, \ldots, t_n)$,
$\overline{a}_2(t_2) :- a_1(t_1), \cdots, a_n(t_n), \sigma(t_1, \ldots, t_n)$,
...
$\overline{a}_n(t_n) :- a_1(t_1), \cdots, a_n(t_n), \sigma(t_1, \ldots, t_n)$.
3- for each atom $a(t)$ occurring in some marked denial constraint, insert into Π_{cqa} a rule: $a^*(t) :- a(t), not\ \ \overline{a}(t)$.
4- for all marked constraints of the form c_2 in C, insert the following rules in Π_{cqa}:
$\overline{\overline{a_1}}(t) :- a_1^*(t), not\ \ aux^r_{a_2(t)}(t).\ \ \ aux^r_{a_2(t)}(t) :- a_2^r(t, t')$.
5- for each $a(t)$ occurring in some marked constraint insert into Π_{cqa} the following rules: $a^r(t) :- a^*(t), not\ \ \overline{a}(t), not\ \ \overline{\overline{a}}(t)$.
6- build Q_{cqa} from Q by replacing each $a(t)$ by $a^r(t)$ whenever $a(t)$ occurs in some marked constraint.

First of all, note that the new algorithm produces only non-redundant rules (i.e. the rules encoding constraints that influence the query answering process). Moreover, it is worth noticing that the rules produced by step 2, corresponding to "easy" constraints are non-disjunctive,[5] while, those produced by step 1, corresponding to "hard" constraints

[5] In the "easy" cases the original database can be repaired by simply removing all the conflicting tuples. This can be done because each repair can be obtained from the original database by removing a single tuple among the ones that violate the same constraint. When rules of this kind are employed the answer sets do not correspond to repairs, but CQA still corresponds to cautious reasoning.

are disjunctive. This is a pay-as-you-go technique where the usage of complex evaluation algorithms is limited to either intractable cases or to cases in which tractability results are not known. Moreover, note that the same query may involve both easy and hard constraints, but disjunctive rules are used only for the hard ones.

For example, suppose that we add to the global schema of our ongoing example a new binary relation *c*(code,name) representing the list of customers, and that code is a key for *c*. Moreover, suppose that we ask for the query Q = c(X,Y), e(X,Y)? retrieving the customers that are also employees of the bank. In this case, the query is quantifier free, and only denial constraints are marked visiting the constraint graph. Indeed, it is easy to see that there is no way to reach m in the constraint graph starting from the query atoms since the arc generated for the inclusion dependency between *m* and *e* goes from *m* to *e*. This means that condition $1.a$ is verified, all marked constraints are "easy", and the produced program is:

$\overline{e}(X,Y) :- e(X,Y), e(X,Z), Y \neq Z.$
$\overline{e}(X,Z) :- e(X,Y), e(X,Z), Y \neq Z.$
$\overline{e}(X,Y) :- e(X,Y), e(Z,Y), X \neq Z.$
$\overline{e}(Z,Y) :- e(X,Y), e(Z,Y), X \neq Z.$
$\overline{c}(X,Y) :- c(X,Y), c(X,Z), Y \neq Z.$
$\overline{c}(X,Z) :- c(X,Y), c(X,Z), Y \neq Z.$
$e^*(t) :- e(t), not\ \ \overline{e}(t).$
$c^*(t) :- c(t), not\ \ \overline{c}(t).$
$e^r(X,Y) :- e(X,Y), not\ \ \overline{e}(X,Y), not\ \ \overline{\overline{e}}(X,Y).$
$c^r(X,Y) :- c(X,Y), not\ \ \overline{c}(X,Y), not\ \ \overline{\overline{c}}(X,Y).$
$c^r(X,Y), e^r(X,Y)?$

Note that the obtained program is non-disjunctive and stratified and it can be evaluated in polynomial time. In this case, the only answer set of the program contains the consistent answers to the original query.

4 The Integration System

The general architecture of the system incorporating the proposed approach is shown in Figure 1. It is intended to simplify both the integration system design and the querying activities by exploiting a user-friendly GUI. Specifically, at design time, the user can:

- Graphically design the global schema and the mappings (which we recall are expressed by UCQs) between global relations and source schemas.
- Specify data transformation rules on source data; these can be implemented by suitable functions defined in the working database as stored functions.
- Specify global constraints, in order to define quality parameters that global integrated data must satisfy.

At query time, the user can exploit a QBE-like interface to express queries over the global schema; these are internally expressed in datalog as UCQs. The "plain" query is then elaborated by the CQA Rewriter which takes into account both mappings and global constraints to express the query over the sources and to handle inconsistencies

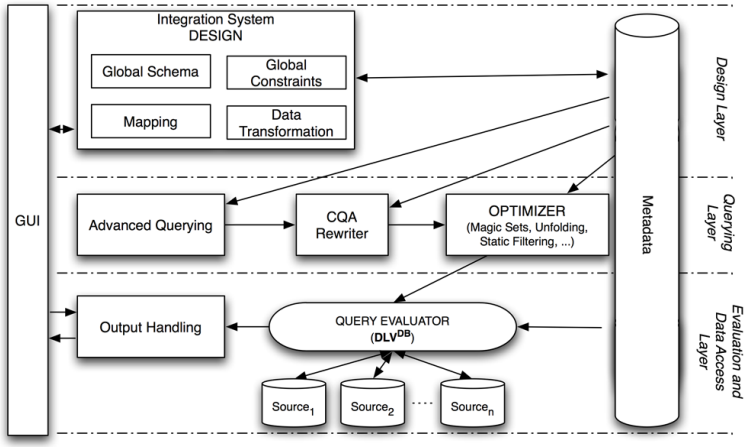

Fig. 1. System Architecture.

possibly involving the query answers; the output of the CQA Rewriter is then a (possibly disjunctive) datalog program which is fed to the Optimizer for further elaboration. The Optimizer applies rewriting strategies which aim at pushing down selections directly onto the sources and at "localizing" over conflicting data as much as possible of the needed reasoning. Finally, the optimized program is fed to the Query Evaluator which executes the grounding phase totally on the DBMS and loads in main-memory only data strictly necessary to resolve conflicts. The output of this evaluation is then the query answer, which is proposed graphically back to the user. More in detail, the Query Evaluator engine of our integration system is DLV^{DB} [7]. It is a DLP evaluator born as a database oriented extension of the well known DLV system [13]. It has been recently extended [8] for dealing with unstratified negation, disjunction and external function calls. The main peculiarities of DLV^{DB} related to the data integration system are: *(i)* it allows the handling of (possibly distributed) large amounts of data stored in autonomous databases; *(ii)* GAV mappings defining the integration system can be directly evaluated on the database where data resides, without further elaboration; *(iii)* it embodies some query-oriented optimization strategies, like magic-sets.

5 Experiments

In this section we present some of the experiments we carried out to asses the effectiveness of our approach to consistent query answering.

5.1 Data Set

We exploited the real-world data integration framework developed in the INFOMIX project (IST-2001-33570) [6] which integrates data from a real university context. In particular, considered data sources were available at the University of Rome "La Sapienza". These comprise information on students, professors, curricula and exams in various

faculties of the university. This data is dispersed over several databases in various (autonomous) administration offices.

There are about 35 data sources in the application scenario, which are mapped into 14 global schema relations with about 20 GAV mappings and 29 integrity constraints. We call this data set **Infomix** in the following.

Besides the original source database instance (which takes about 16Mb on DBMS), we obtained bigger instances artificially. Specifically, we generated a number of copies of the original database; each copy is disjoint from the other ones but maintains the same data correlations between instances as the original database. This has been carried out by mapping each original attribute value to a new value having a copy-specific prefix.

Then, we considered two further datasets, namely **Infomix-x-10** and **Infomix-x-50** storing 10 copies (for a total amount of 160Mb of data) and 50 copies (800Mb) of the original database, respectively; clearly, in both cases one of the copies is the original database itself.

5.2 Tested Queries

As previously pointed out, standard rewriting for CQA makes the time complexity of query evaluation to be in co-NP in most cases; however, our optimization allows in many relevant cases to simplify the rewriting in such a way that the complexity of the evaluation of the corresponding program can be in P.

In order to carry out a comprehensive performance analysis, we designed a set of queries spanning over the following perspectives:

- As for the computational complexity perspective we designed queries whose:
  - evaluation complexity with standard rewriting stays in co-NP and evaluation complexity with optimized rewriting stays in P;
  - evaluation complexity with standard rewriting stays in co-NP and evaluation complexity with optimized rewriting remains in co-NP;
- As for the constraints perspective we designed queries involving:
  - Arbitrary Denial constraints only (D in the following)
  - Key constraints only (K in the following)
  - Inclusion dependencies only (I in the following)
  - Arbitrary Denial and Inclusion dependencies (D+I in the following)
  - Key constraints and Inclusion dependencies (K+I in the following)
- As for the query class perspective we designed:
  - Unrestricted Conjunctive queries (UC in the following)
  - Quantifier-free queries (QF in the following)
  - Simple Conjunctive queries (SC in the following)
- As for the query design perspective we considered:
  - queries with different arities (i.e. different number columns in the result)[6]
  - queries with and without constants

We designed and ran several queries. Table 1 summarizes the characteristics of a representative set of them. Here *Number of source tuples* indicates the number of tuples of all source relations involved by the query.

[6] An arity equal to 0 indicates that the query asks only if some assertion is true or false in the database.

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
Optimized Query Evaluation	co-NP	co-NP	P	P	P	P
Query Class	QF	SC	SC	QF	SC	SC
Involved Constraints	K	D	D	D	K	K
Query arity	7	1	0	0	1	2
N. of source tuples						
infomix	45575	37546	20575	67272	204	61723
infomix-x-10	455750	375460	205750	672720	2040	617230
infomix-x-50	2278750	1877300	1028750	3363600	10200	3086150

	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}
Optimized Query Evaluation	P	co-NP	P	co-NP	P	P
Query Class	SC	SC	SC	UC	QF	SC
Involved Constraints	K	K+I	K	K+I	I	D+I
Query arity	2	3	6	2	3	0
N. of source tuples						
infomix	104818	17266	16148	3749	17725	37831
infomix-x-10	1048180	172660	161480	37490	177250	378310
infomix-x-50	5240900	863300	807400	1873950	886250	1891550

Table 1. Summary of tested queries.

5.3 Compared Methods

In order to asses the characteristics of the proposed optimizations, we measured the execution time of each query with both the standard and our optimized rewriting. Moreover, since the magic sets technique has been recently extended to support also disjunctive programs [15], we considered interesting to evaluate execution times of both rewritings with the addition of magic sets on them; query rewriting for CQA and magic sets have been applied in cascade.

Note, however, that the magic sets technique can be applied only on queries involving constants; indeed, the aim of the technique is to "push down" constants in the query onto source relations, thus allowing to reduce the amount of data to reason about. Moreover, the magic sets method may add several rules (and possibly unstratified negation) to the original program, thus introducing some overhead in the computation.

Our intuition is that magic sets optimization and our optimizations are complementary and, consequently, their benefits may be summed up if the query involves some constant. It is also interesting to evaluate the impact of the overhead introduced by the approaches on the overall response time.

Summarizing, we tested four methods: *(i)* Standard Rewriting, *(ii)* Optimized Rewriting, *(iii)* Standard Rewriting with Magic Sets, *(iv)* Optimized Rewriting with Magic Sets.[7]

[7] Magic sets have been tested only on queries Q_1, Q_2, Q_3, and Q_4, since the other queries are constant-free.

5.4 Results and discussion

All tests have been carried out on an Intel Core 2 Duo T7300, 2.0 GHz, with 2 Gb Ram, running Windows 7 Operating System. We set a time limit of 30 minutes after which query execution has been killed. Results obtained for tested queries (showing times in seconds) are illustrated in Figure 2. The bar for a method is absent in the graphs if query answering time was higher than the limit[8].

From the analysis of the figures and the characteristics of the queries reported in Table 1, we may draw the following observations: The optimized rewriting almost always provides important improvements in query performance. The only exception is for query Q_{11} for which no optimization was possible and, consequently, standard and optimized rewritings coincide. Performance improvements of the optimized rewriting w.r.t. the standard one have been registered up to 86%[9] with a quantifier free query over denial constraints. Note also that the best times are always registered when the proposed optimization is active.

Our intuition about the "additivity" of the magic sets over our optimization has been confirmed by experimental results. In fact, the application of magic sets always improves its performance, at least on big data sets. As for the smallest data set, the overhead introduced by magic sets is sensible and the (initially small) response time increases in some cases. It is interesting to observe that the application of our optimization *and* the magic sets allowed performance improvements up to 95% w.r.t. the standard rewriting.

Finally, it is worth pointing out that the scaling of the optimized algorithm over the three data sets is generally better than the standard one.

6 Conclusion and Ongoing Work

In this paper we presented an approach that allows to efficiently handle consistent query answering under a wide variety of integrity constraints. The effectiveness of the approach is obtained by the assumption of complete sources and an optimized algorithm which is capable to identify both tractable queries and portions of the queries that may be treated efficiently. The approach is part of a complete system for data integration based on ASP whose query evaluator engine allows to carry out querying directly on the databases where data reside even in an ASP context. Results of our experimental activity demonstrate the effectiveness of the approach. As far as ongoing work, we are investigating for further optimizations that can be included in the algorithm to further improve query answering performances.

References

1. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. PODS 2002. (2002) 233–246

[8] Recall that Magic Sets have been applied only on Q_1-Q_4 and therefore results for this method are reported on the first four graphs only.

[9] This information has been computed over the unhalted queries only.

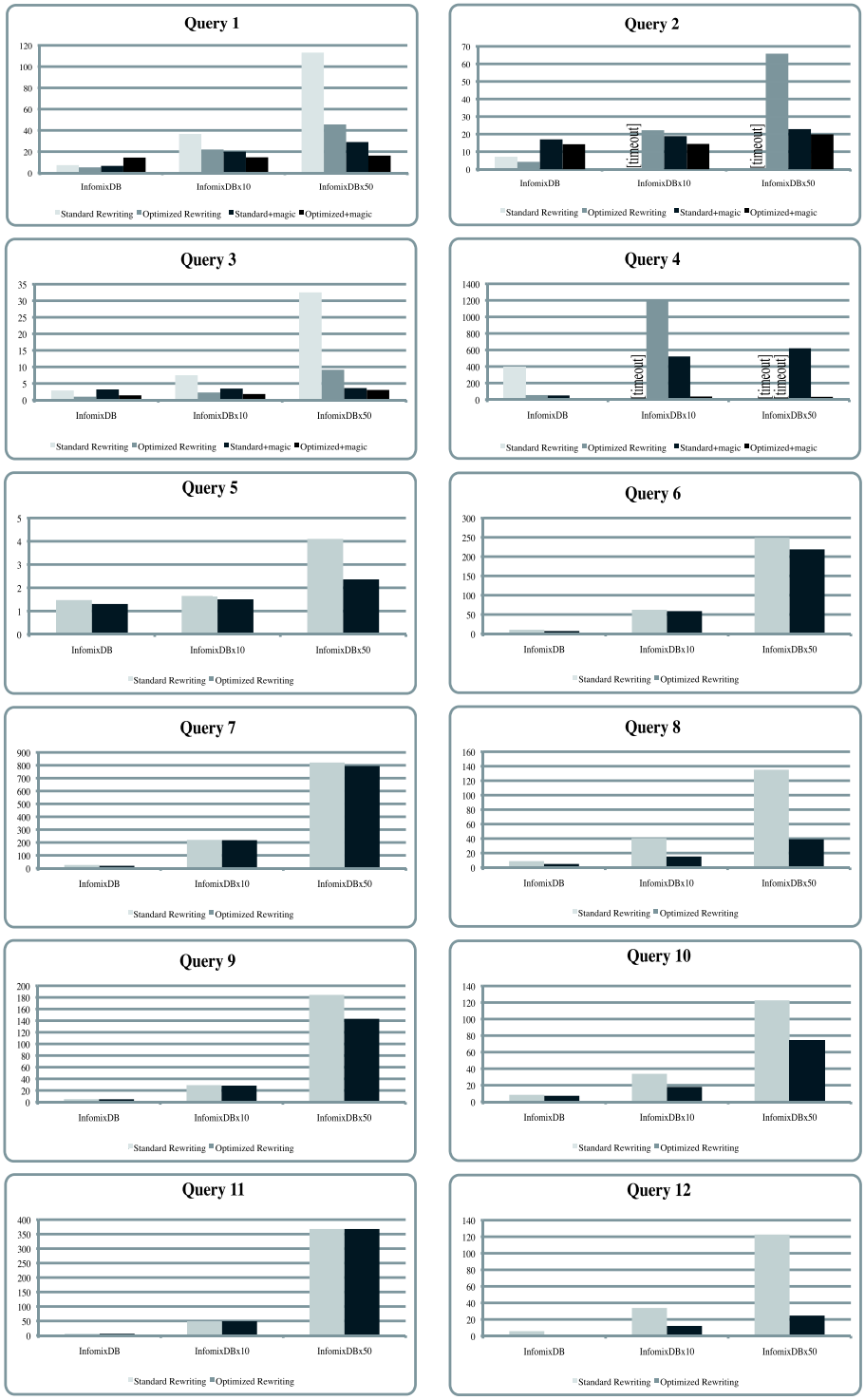

Fig. 2. Query Evaluation Execution Times.

2. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent Query Answers in Inconsistent Databases. In: Proc. PODS 1999, ACM Press (1999) 68–79
3. Bertossi, L.E., Hunter, A., Schaub, T., eds.: Inconsistency Tolerance. Volume 3300 of LNCS. Springer (2005)
4. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Information and Computation **197**(1–2) (2005) 90–121
5. Calì, A., Lembo D., ROsati R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In Proc. of PODS'03, ACM (2003) 260–271
6. Leone, N. et al.: The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data In: Proc. ACM SIGMOD 2005. (2005) 915–917
7. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. Theory and Practice of Logic Programming (TPLP) . Cambridge University Press, UK. **8(2)** (2008) 129–165
8. Terracina, G., De Francesco, E., Panetta, C., Leone, N.: Enhancing a DLP system for advanced database applications. In: Proc. of RR'08, LNCS, Springer (2008) 119–134
9. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Logic Programming: Proceedings Fifth Intl Conference and Symposium, Cambridge, Mass., MIT Press (1988) 1070–1080
10. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. **9** (1991) 365–385
11. Minker, J.: On Indefinite Data Bases and the Closed World Assumption. In Loveland, D.W., ed.: Proceedings 6^{th} Conference on Automated Deduction (CADE '82). Volume 138., New York, Springer (1982) 292–308
12. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Transactions on Database Systems **22**(3) (September 1997) 364–418
13. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Trans. Comput. Log. **7(3)** (2006) 499–562
14. Halevy, A.Y.: Data integration: A status report. In: 10th Conference on Database Systems for Business, Technology and Web (BTW 2003). (2003) 24–29
15. Cumbo, C., Faber, W., Greco, G., Leone, N.: Enhancing the magic-set method for disjunctive datalog programs. In: Proceedings of the the 20th International Conference on Logic Programming – ICLP'04. Volume 3132. (2004) 371–385