Practical Query Answering in Data Exchange Under Inconsistency-Tolerant Semantics

Balder ten Cate
UC Santa Cruz
Google
btencate@ucsc.edu

Richard L. Halpert
UC Santa Cruz
rhalpert@ucsc.edu

Phokion G. Kolaitis
UC Santa Cruz
IBM Research - Almaden
kolaitis@ucsc.edu

ABSTRACT

Exchange-repair semantics (or, XR-Certain semantics) is a recently proposed inconsistency-tolerant semantics in the context of data exchange. This semantics makes it possible to provide meaningful answers to target queries in cases in which a given source instance cannot be transformed into a target instance satisfying the constraints of the data exchange specification. It is known that computing the answers to conjunctive queries under XR-Certain semantics is a coNP-complete problem in data complexity. Moreover, this problem can be reduced in a natural way to cautious reasoning over stable models of a disjunctive logic program.

Here, we explore how to effectively perform XR-Certain query answering for practical data exchange settings by leveraging modern sophisticated solvers for disjunctive logic programming. We first present a new reduction, accompanied by an optimized implementation, of XR-Certain query answering to disjunctive logic programming. We then evaluate this approach on a benchmark that we introduce here and which is modeled after a practical data exchange problem in computational genomics. Specifically, we present a benchmark scenario that mimicks a portion of the UCSC Genome Browser data import process. Our initial results, based on real genomic data, suggest that the solvers we apply fail to take advantage of some critical exploitable structural properties of the specific instances at hand. We then develop an improved encoding to take advantage of these properties using techniques inspired by the notion of a repair envelope. The improved implementation utilizing these techniques computes query answers ten to one thousand times faster for large instances, and exhibits promising scalability with respect to the size of instances and the rate of target constraint violations.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems—*relational databases, rule based databases, query processing*

EDBT 2016 Bordeaux, France

General Terms

Theory, Algorithms, Design

Keywords

Data exchange, certain answers, repairs, consistent query answering, disjunctive logic programming, stable models

1. INTRODUCTION

Data exchange is the task of transforming data structured under a source schema into data structured under a target schema in such a way that all constraints in a fixed set of source-to-target constraints and in a fixed set of target constraints are satisfied. During the past decade, there has been an extensive and multifaceted investigation of data exchange (see the monograph [1]). There are two main algorithmic problems in data exchange: the problem of materializing an instance that, together with a given source instance satisfies all constraints (such an instance is a called a *solution* of the given source instance) and the problem of computing the *certain answers* to a query over the target schema, i.e., the intersection of the answers to the query over all solutions of a given source instance. In data exchange settings with a non-empty set of target constraints, it frequently happens that a given source instance has no solution. In particular, this may happen when the source instance at hand contains inconsistencies or conflicting information that is exposed by the target constraints. The standard data exchange frameworks are not able to provide meaningful answers to target queries in such circumstances; in fact, the certain answers to every target query trivialize. To address this problem and to give meaningful answers to target queries, we recently introduced the framework of *exchange-repair* semantics (or, XR-Certain semantics) [8]. This is an inconsistency-tolerant framework that is based on the notion of *source* repairs, where, informally, a source repair is a source instance that differs minimally from the original source data, but has a solution. In turn, source repairs give rise to the notion of the XR-Certain *answers* to target queries, which, by definition, are the intersection of the answers to the query over all solutions of all source repairs of the given source instance. It should be noted that inconsistency-tolerant semantics have also been investigated in the context of data integration (see, e.g., [7, 17]) and in the context of ontology-based data access (OBDA) (see, e.g., the recent survey [5]). In [9], which is the full version of [8], we provided a detailed comparison between the XR-Certain semantics and the inconsistency-tolerant semantics in these two other frameworks. In partic-

ular, we showed that, as regards consistent query answering, the exchange-repairs framework and the OBDA framework can simulate each other.

A data exchange task is specified using a *schema mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$, where $\mathbf{S}$ is the source schema, $\mathbf{T}$ is the target schema, Σ_{st} is a set of constraints between $\mathbf{S}$ and $\mathbf{T}$, and Σ_{t} is a set of constraints on $\mathbf{T}$. The most extensively studied schema mappings are the ones in which Σ_{st} is a set of source-to-target tuple-generating dependencies (s-t tgds) and Σ_{t} is a weakly-acyclic set of target tuple-generating dependencies (target tgds) and target equality-generating dependencies (target egds) [11]. Note that tuple-generating dependencies are also known as GLAV (global-and-local as view) constraints; as special cases, they contain the classes of GAV (global-as-view) constraints and LAV (local-as-view) constraints. In [8], it was shown that computing the XR-Certain answers to target conjunctive queries is a coNP-complete problem in data complexity; in fact, this intractability persists even in the case in which Σ_{st} is a set of GAV constraints and Σ_t is a set of egds. Moreover, a connection with disjunctive logic programming was unveiled in [8] by showing that the XR-Certain answers of conjunctive queries can be rewritten as the cautious answers of a union of conjunctive queries with respect to the stable models of a disjunctive logic program over a suitably defined expansion of the source schema.

Here, our main aim is to explore how to effectively perform XR-Certain query answering for practical data exchange settings by leveraging modern sophisticated solvers for disjunctive logic programming. Disjunctive logic programming is a suitable formalism for coping with the intractability of XR-Certain query answering because it goes beyond the natural expressiveness of SQL while still remaining in a declarative framework. Our first technical result is a new improved reduction, accompanied by an optimized implementation, of the problem of computing the XR-Certain answers of queries in the context of data exchange to the problem of computing the certain answers of queries with respect to the stable models of disjunctive logic programs. We then evaluate this approach on a benchmark that we introduce here and which is modeled after a practical data exchange problem in computational genomics. Specifically, we present a benchmark scenario that mimics a portion of the data import process of the UCSC Genome Browser (`https://genome.ucsc.edu/`), a widely used genomics resource that "contains the reference sequence and working draft assemblies for a large collection of genomes." We believe that data sets from computational sciences, such as computational genomics, are particularly in need of concepts and techniques that, like XR-Certain answers, eliminate the unquantifiable uncertainty that arise from constraint violations.

We carry out two experimental evaluations using real genomic data. The first is based on what we call a *monolithic* approach, which generates a disjunctive logic program from a given query and source instance, and then runs the *clingo* solver from the Potassco collection [12]. The results of this evaluation suggest that the solver fails to take advantage of some critical exploitable structural properties of the specific instances at hand. Intuitively, the cost of transforming the data from the source schema into the target schema is embedded in the execution cost of running each individual query, which causes large instances to become unworkable even for simple queries. In view of this, we develop a different *segmentary* approach that utilizes an improved encoding to take advantage of the aforementioned structural properties using techniques inspired by the notion of a *repair envelope*. The improved implementation utilizing these techniques computes query answers ten to one thousand times faster than the monolithic approach for large instances, and exhibits promising scalability with respect to the size of instances and the rate of target constraint violations.

2. PRELIMINARIES

This section contains definitions of basic notions and a minimum amount of background material on data exchange and on disjunctive logic programming. More detailed information about schema mappings and certain answers can be found in [1, 11].

Instances, Queries, and Homomorphisms. Fix an infinite set Const of elements, and an infinite set Nulls of elements such that Const and Nulls are disjoint. A *schema* $\mathbf{R}$ is a finite set of relation symbols, each having a designated arity. An $\mathbf{R}$-*instance* is a finite database I over the schema $\mathbf{R}$ whose active domain is a subset of $\text{Const} \cup \text{Nulls}$. A *fact* of an $\mathbf{R}$-instance I is an expression of the form $R(a_1, \ldots, a_k)$, where R is a relation symbol of arity k in $\mathbf{R}$ and $(a_1, \ldots, a_k)$ is a member of the relation R^I on I that interprets the symbol R. Every $\mathbf{R}$-instance can be identified with the set of its facts. We say that an $\mathbf{R}$-instance I' is a *sub-instance* of an $\mathbf{R}$-instance I if $I' \subseteq I$, where I' and I are viewed as sets of facts. If I is an $\mathbf{R}$-instance and $\mathbf{R}' \subseteq \mathbf{R}$, then by the $\mathbf{R}'$-*restriction* of I we will mean the subinstance of I containing only those facts that involve relations from $\mathbf{R}'$.

We assume familiarity with *conjunctive queries* (CQs) and *unions of conjunctive queries* (UCQs). The answers to a query q in an instance I are denoted by $q(I)$, and we denote by $q{\downarrow}(I)$ the answers of q on I that contain only values from Const.

The *active domain* of an instance I is the set of values from $\text{Const} \cup \text{Nulls}$ that occur in facts of I. By a *homomorphism* from an $\mathbf{R}$-instance I to another $\mathbf{R}$-instance I', we mean a map h from the active domain of I to the active domain of I', such that $h(c) = c$ for all $c \in \text{Const}$, and such that for every fact $\text{R}(v_1, ..., v_n) \in I$ we have that $\text{R}(h(v_1), \ldots, h(v_n)) \in I'$.

Schema Mappings. A *tuple-generating dependency (tgd)* over a schema $\mathbf{R}$ is an expression of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$, where $\phi(\mathbf{x})$ and $\psi(\mathbf{x}, \mathbf{y})$ are conjunctions of atoms over $\mathbf{R}$. Tgds are also known as GLAV (global-and-local-as-view) constraints. Two important special cases are the GAV constraints and the LAV constraints. A GAV constraint is a tgd of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow P(\mathbf{x}))$ (that is, the right-hand side of the implication consists of a single atom without existential quantifiers) and a LAV constraint is a tgd of the form $\forall \mathbf{x}(R(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$ (that is, the left-hand side of the implication consists of a single atom).

Let $\mathbf{S}$ and $\mathbf{T}$ be disjoint schemas, called the *source* schema and the *target* schema. A *source-to-target tgd* (*s-t tgd*, or, source-to-target GLAV constraint) is a tgd as defined above, where $\phi(\mathbf{x})$ is a conjunction of atoms over $\mathbf{S}$ and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over $\mathbf{T}$.

An *equality-generating dependency (egd)* over a schema $\mathbf{R}$ is an expression of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow x_i = x_j)$ with $\phi(\mathbf{x})$ a conjunction of atoms over $\mathbf{R}$.

For the sake of readability, we will frequently drop universal quantifiers when writing tgds and egds.

A *schema mapping* is a quadruple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$, where $\mathbf{S}$ is a source schema, $\mathbf{T}$ is a target schema, Σ_{st} is a finite set of s-t tgds, and Σ_{t} is a finite set of tgds and/or egds over the target schema. We will also call such schema mappings GLAV+(GLAV, EGD) schema mappings. In the special case where Σ_{st} consists of (source-to-target) GAV constraints and Σ_{t} consists of GAV constraints and/or egds, we will say that $\mathcal{M}$ is a GAV+(GAV, EGD) schema mapping.

Universal Solutions and Certain Answers. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ be a schema mapping, and let I be a source instance. As usual in data exchange, we will assume that the source instances we consider do not contain null values.

A target instance J is a *solution* for a source instance I w.r.t. $\mathcal{M}$ if the pair (I, J) satisfies the constraints of $\mathcal{M}$, that is, I and J together satisfy Σ_{st}, while J satisfies Σ_{t}. In general, a source instance may have many solutions. A *universal* solution for I (with respect to $\mathcal{M}$) is a solution J for I such that for all solutions J' of I, there is a homomorphism h from J to J'. Universal solutions are considered the preferred solutions in data exchange. One reason for this is that universal solutions can be used to compute certain answers to target queries.

If q is a query over the target schema $\mathbf{T}$, then the *certain answers* of q with respect to I and $\mathcal{M}$ are defined as

$$\text{certain}(q, I, \mathcal{M}) = \bigcap\{q(J) : \ J \text{ is a solution for } I \text{ w.r.t. } \mathcal{M}\}$$

It was shown in [11] that, if J is a universal solution for a source instance I w.r.t. a schema mapping $\mathcal{M}$, then for every conjunctive query q, it holds that $\text{certain}(q, I, \mathcal{M}) = q{\downarrow}(J)$.

Weak Acyclicity and the Chase. If $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ is an arbitrary GLAV+(GLAV, EGD) schema mapping, then a given source instance may have no solution or it may have a solution, but no universal solution. For this reason, in [11] the concept of *weak acyclicity* was introduced, and it was shown that, when Σ_{t} is the union of a *weakly acyclic* set of target tgds and a set of egds, then, for all source instances I, a solution exists if and only if a universal solution exists. Moreover, the *chase procedure* can be used to determine in polynomial time (data complexity) whether a solution for I exists and, if so, to construct a universal solution for I in time polynomial in the size of I. The obtained solution, which we will denote by $chase(I, \mathcal{M})$ (when it exists), is known as the *canonical universal solution* of I. We refer to [11] for more details, including the definition of weak acyclicity and of the chase procedure.

By a GLAV+(WA-GLAV, EGD) schema mapping we will mean a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$, where Σ_{st} is a finite set of s-t tgds, and Σ_{t} is the union of a weakly acyclic set of target tgds and a set of egds. We spell out here two basic facts that are used in several arguments in this paper: let $\mathcal{M}$ be any GLAV+WA-GLAV schema mapping (without egds). Then (i) every source instance I has a solution (and hence has a canonical universal solution), and (ii) whenever $I' \subseteq I$, then $chase(I', \mathcal{M}) \subseteq chase(I, \mathcal{M})$. The latter is also known as the *monotonicity of the chase*.

Disjunctive Logic Programming. A *disjunctive logic program* (DLP program) Π over a schema $\mathbf{R}$ is a finite collection of rules of the form

$$\alpha_1 \vee \ldots \vee \alpha_n \leftarrow \beta_1, \ldots, \beta_m, \neg\gamma_1, \ldots, \neg\gamma_k.$$

where $n, m, k \geq 0$ and $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m, \gamma_1, \ldots, \gamma_k$ are atoms formed from the relations in $\mathbf{R} \cup \{=, \neq\}$, using the constants in Const and first-order variables. A DLP program is said to be *ground* if it consists of rules that do not contain any first-order variables. A *model* of Π is an $\mathbf{R}$-instance I over the domain Const that satisfies all rules of Π (viewed as universally quantified first-order sentences). A *minimal model* of Π is a model M of Π such that there does not exist a model M' of Π where the facts of M' form a strict subset of the facts of M. For a ground DLP Π over a schema $\mathbf{R}$ and an $\mathbf{R}$-instance M over the domain Const, the *reduct* Π^M of Π with respect to M is the DLP containing, for each rule $\alpha_1 \vee \ldots \vee \alpha_n \leftarrow \beta_1, \ldots, \beta_m, \neg\gamma_1, \ldots, \neg\gamma_k$, with $M \not\models \gamma_i$ for all $i \leq k$, the rule $\alpha_1 \vee \ldots \vee \alpha_n \leftarrow \beta_1, \ldots, \beta_m$. A *stable model* of a ground DLP Π is an $\mathbf{R}$-instance M over the domain *Const* such that M is a minimal model of the reduct Π^M. See [13] for more details. The *cautious answers* to a query q, w.r.t. a DLP program Π (under the stable model semantics) are defined as

$$\bigcap\{q(s) \mid s \text{ is a stable model of } \Pi\} \ .$$

The stable model semantics is the most widely used semantics of DLP programs, and many solvers have been developed that support reasoning over stable models. In particular, stable models of disjunctive logic programs have been well-studied as a way to compute database repairs ([18] provides a thorough treatment).

3. EXCHANGE REPAIR FRAMEWORK

We briefly recall here the exchange-repair framework that was introduced in [8]. The development of this framework was motivated by the observation that the definition of $\text{certain}(q, I, \mathcal{M})$ trivializes when a source instance I has no solution w.r.t. a given schema mapping $\mathcal{M}$. XR-Certain answers were proposed as a semantics that provides meaningful answers to queries in such cases.

Definition 1. [8] Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ be a schema mapping, let I be an $\mathbf{S}$-instance.

1. A source instance I' is said to be a *source repair* of I with respect to $\mathcal{M}$ if $I' \subseteq I$, I' has a solution with respect to $\mathcal{M}$, and no instance I'' with $I' \subsetneq I'' \subseteq I$ has a solution with respect to $\mathcal{M}$.

2. We say that a pair (I', J') is an *exchange-repair solution* (or XR-Solution) for I with respect to $\mathcal{M}$ if I' is a source repair of I with respect to $\mathcal{M}$ and J' is a solution for I' with respect to $\mathcal{M}$.

3. For a query q over the target schema $\mathbf{T}$, the *XR-certain answers* to q in I w.r.t. $\mathcal{M}$ is the set

$$\text{XR-Certain}(q, I, \mathcal{M}) \ = \ \bigcap\{q(J') \mid (I', J') \text{ is an XR-Solution for } I \text{ w.r.t. } \mathcal{M}\}.$$

Note that, whenever a source instance I *does* have solutions w.r.t. $\mathcal{M}$, then, for all queries q, we have that

XR-Certain$(q, I, \mathcal{M})$ = certain$(q, I, \mathcal{M})$. While the XR-Certain semantics takes inspiration from the well-established notions of *database repairs* and *consistent query answers* [2, 4], the precise definition of the semantics reflects important assumptions that are specific to the context of data exchange. Specifically, the definitions of XR-Solution and XR-Certain reflect the fact that in a data exchange setting, it is preferred to make tgds satisfied by deriving additional facts, rather than by deleting facts; and the data used to answer target queries should derive from coherent sets of source facts.

It was shown in [8] that, in the case of GLAV+(WA-GLAV, EGD) schema mappings, the data complexity of XR-Certain query answering for conjunctive queries is coNP-complete (note that the restriction to weakly acyclic schema mappings is necessary here, since it follows from results in [11] that the same problem is undecidable for arbitrary GLAV+(GLAV, EGD) schema mappings). Furthermore, several approaches to query answering were studied in [8]. In particular, it was shown that, for GLAV+(WA-GLAV, EGD) schema mappings, XR-certain answers can be computed by means of a disjunctive logic program. We discuss this approach in the next section.

4. BASIC APPROACH TO QUERY ANSWERING

Based on the initial results in [8], we pursue here the development of a practical system for XR-Certain query answering via disjunctive logic programming, leveraging modern sophisticated solvers. Note that the emergence of such powerful solvers for NP-hard problems has already enabled practical solutions for many other computationally hard problems in industry. Concretely, in this section, we present a first implementation of XR-Certain query answering based on a translation along the lines of [8], that takes as input a schema mapping $\mathcal{M}$ and a source instance I, and produces a single, typically large, DLP program whose stable models describe the XR-solutions of I w.r.t. $\mathcal{M}$. We refer to this as the *monolithic approach*, in order to contrast it with another approach, which will be presented in Section 6, involving multiple DLP programs of smaller size.

The first step in this approach consists of a reduction from the general case of GLAV+(WA-GLAV, EGD) schema mappings to the case of GAV+(GAV, EGD) schema mappings.

THEOREM 1 ([8]). *If $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ is a* GLAV+(WA-GLAV, EGD) *schema mapping and q a conjunctive query over $\mathbf{T}$, then there exist a* GAV+(GAV, EGD) *schema mapping $\hat{\mathcal{M}}$ and union of conjunctive queries $\hat{q}$ such that* XR-Certain$(q, I, \mathcal{M})$ = XR-Certain$(\hat{q}, I, \hat{\mathcal{M}})$.

The resulting schema mapping may in general be exponentially larger than the original. However, as we will see in Section 5.2, our implementation incurs only a modest increase when applied to our benchmark.

Next, in [8], we present a very natural and concise encoding of XR-Certain for GAV+(GAV, EGD) schema mappings as the cautious answers over the parallel circumscription of a disjunctive logic program Π. In [15] it was shown that the problem of finding such models can be subsequently reduced to that of computing stable models of a translation of Π into a new disjunctive logic program. However, the translation involved requires the explicit representation of the herbrand base of the source instance, which can be prohibitively large even for small source instances. We now present an improved, direct reduction of XR-Certain to the cautious answers over stable models of a disjunctive logic program.

In order to facilitate the discussion below, it is convenient to introduce the notion of a *canonical* XR-Solution. An XR-Solution (I', J') for I w.r.t. $\mathcal{M}$ is said to be a *canonical* XR-Solution if J' is a canonical universal solution for I' w.r.t. $\mathcal{M}$, that is, $J' = chase(I, \mathcal{M})$.

For any GAV+(GAV, EGD) schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$, let $\Pi_{\mathcal{M}}$ be the DLP program given in Figure 1. Observe that the schema of the program $\Pi_{\mathcal{M}}$ contains multiple distinct copies of each relation from $\mathbf{S} \cup \mathbf{T}$. The program in Theorem 2 intends to describe the canonical XR-Solutions of a source instance. Suppose (I', J') is a canonical XR-Solution for some source instance I w.r.t. a GAV+(GAV, EGD) schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$, and let J be the canonical universal solution for I w.r.t. the tgds of $\mathcal{M}$. The program introduces three predicates for each source relation, one with the original name meant to contain the facts of I, one subscripted with `d` ("deleted") meant to contain the facts of $I \setminus I'$, and one subscripted with `r` ("remains") meant to contain the facts of I'. The program also introduces these same three predicates for each target relation, plus a fourth subscripted with `i` ("incidentally deleted") meant to contain the target facts in $J \setminus J'$ that are also contained in some subset $J'' \supseteq J'$ of J that is consistent with Σ_{t} (that is, they are not in a canonical XR-Solution but they may appear in some other XR-Solution).

For every stable model M of $\Pi_{\mathcal{M}}$, we denote by I^M and J^M the $\mathbf{S}$-instance and $\mathbf{T}$-instance consisting of those facts $R(\mathbf{a})$ in the relevant schema for which $R_r(\mathbf{a}) \in M$.

THEOREM 2. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a* GAV+(GAV, EGD) *schema mapping and I a source instance. The XR-solutions of I w.r.t. $\mathcal{M}$ are precisely those pairs of instances that are of the form (I^M, J^M) for some stable model M of $\Pi_{\mathcal{M}} \cup I$.*

COROLLARY 1. *Let $\mathcal{M}$ be a* GAV+(GAV, EGD) *schema mapping. For every union of conjunctive queries q and for every source instance I,*

$$\text{XR-Certain}(q, I, \mathcal{M}) = \bigcap \{q_r(s) \mid s \text{ a stable model of } \Pi_{\mathcal{M}} \cup I\}$$

where q_r is the query formed from q by replacing every occurrence of a relation R with R_r.

With this new encoding, we can now compute XR-Certain using the disjunctive logic program $\Pi_{\mathcal{M}} \cup I$, whose size is linear in the combined size of the source instance and schema mapping. The DLP generated using the prior approach of [8], in contrast, is lower-bounded in size by $|adom(I)|^s$, where s is the maximum arity of the relations in the source schema.

5. SCENARIO: GENOME BROWSER

Scientific data is a potentially important application of XR-Certain query answering because research decisions and discoveries are often made based on data drawn from a variety of sources, and because unquantifiable uncertainty must

For each tgd $(\mathrm{R1}(\mathbf{x},\mathbf{y}) \wedge ... \wedge \mathrm{R}n(\mathbf{x},\mathbf{y}) \rightarrow \mathrm{T}(\mathbf{x})) \in (\Sigma_{\mathrm{st}} \cup \Sigma_{\mathrm{t}})$, construct a *chase rule*, *deletion rule*, and *target remainder rule* respectively:

$$\begin{array}{lll} \mathtt{T}(\mathbf{x}) & \leftarrow & \mathtt{R1}(\mathbf{x},\mathbf{y}), ..., \mathtt{R}n(\mathbf{x},\mathbf{y}). \\ \mathtt{R1_d}(\mathbf{x},\mathbf{y}) \vee ... \vee \mathtt{R}n_\mathtt{d}(\mathbf{x},\mathbf{y}) & \leftarrow & \mathtt{T_d}(\mathbf{x}), \mathtt{R1}(\mathbf{x},\mathbf{y}), ..., \mathtt{R}n(\mathbf{x},\mathbf{y}), \\ & & \neg\mathtt{R1_i}(\mathbf{x},\mathbf{y}), ..., \neg\mathtt{R}n_\mathtt{i}(\mathbf{x},\mathbf{y}). \\ \mathtt{T_r}(\mathbf{x}) & \leftarrow & \mathtt{R1_r}(\mathbf{x},\mathbf{y}), \ldots, \mathtt{R}n_\mathtt{r}(\mathbf{x},\mathbf{y}). \end{array}$$

For each egd $(\mathrm{R1}(\mathbf{x}) \wedge ... \wedge \mathrm{R}n(\mathbf{x}) \rightarrow x_i = x_j) \in \Sigma_{\mathrm{t}}$, construct a *deletion rule*:

$$\begin{array}{lll} \mathtt{R1_d}(\mathbf{x}) \vee ... \vee \mathtt{R}n_\mathtt{d}(\mathbf{x}) & \leftarrow & \mathtt{R1}(\mathbf{x}), \ldots, \mathtt{R}n(\mathbf{x}), x_i \neq x_j, \\ & & \neg\mathtt{R1_i}(\mathbf{x}), ..., \neg\mathtt{R}n_\mathtt{i}(\mathbf{x}). \end{array}$$

For each relation $\mathrm{R} \in \mathbf{S}$, construct a *source remainder rule*:

$$\mathtt{R_r}(\mathbf{x}) \quad \leftarrow \quad \mathtt{R}(\mathbf{x}), \neg\mathtt{R_d}(\mathbf{x})$$

For each relation $\mathrm{R} \in \mathbf{T}$, construct an *incidental deletion rule*, and the *one-of-three rules*:

$$\begin{array}{lll} \mathtt{R_i}(\mathbf{x}) & \leftarrow & \mathtt{R}(\mathbf{x}), \neg\mathtt{R_r}(\mathbf{x}), \neg\mathtt{R_d}(\mathbf{x}) \\ \bot & \leftarrow & \mathtt{R_r}(\mathbf{x}), \mathtt{R_d}(\mathbf{x}) \\ \bot & \leftarrow & \mathtt{R_r}(\mathbf{x}), \mathtt{R_i}(\mathbf{x}) \\ \bot & \leftarrow & \mathtt{R_d}(\mathbf{x}), \mathtt{R_i}(\mathbf{x}) \end{array}$$

Figure 1: Procedure to construct the disjunctive logic program $\Pi_{\mathcal{M}}$

be eliminated from the data before a decision or discovery is made. The guarantee given by XR-Certain query answers – that all possible repairs of the source instance agree on them – is a natural fit for these circumstances.

Inspired by the UCSC Genome Browser, we present a benchmark that uses real data: a loose simulation of the genome browser data import process. The UCSC Genome Browser database is constructed using a variety of algorithms and public data sources. Our benchmark focuses on the human gene model, a set of genomic sequences which putatively capture the portion of the human genome that encodes proteins. The UCSC Genome Browser algorithms compute these sequences from a reference genome by computing alignments for known/observed proteins and transcripts from the UniProt and GenBank databases [14]. For our purposes, we treat the set of transcripts as given (that is, as a source instance rather than the result of a computation), and we provide a schema mapping mimicking how this data, plus a significant volume of data from the RefSeq, Entrez Gene, and UniProt databases, are combined and transformed into the UCSC Genome Browser database. Our schema mapping makes several loose approximations of scientific reality which serve to maximize the portion of the genome browser schema that we populate, but which also introduce some inconsistency to the data. It is for this reason that we say our schema mapping merely *mimicks* the true UCSC Genome Browser data import process, even though our target schema is faithful to the real database.

The RefSeq and EntrezGene databases are not available for download in a flat relational format. The RefSeq database is offered in a text file format, within which the data are arranged in a nested fashion with transcripts as the top-level elements. Every transcript has associated source and reference information (documenting how and by whom it was observed), and each may have subsections specifying what protein it encodes and what gene it is transcribed from. These subsections often link to other databases using external protein and gene identifiers. Our ETL step places transcripts, sources, references, genes, and proteins into five separate respective relations, all keyed by transcript accession identifier. The EntrezGene database is available in ASN.1 format, which we first convert to xml using the `gene2xml` tool from the NCBI ToolBox [19]. From the resulting xml, we extract the desired fields into a single table using the `xtract` tool from NCBI Entrez Direct [20].

Table 1: Source Instances

Database	Relations	total # of Attributes	total # of Tuples
UCSC*	2	13	165,920
RefSeq	5	38	706,923
EntrezGene	1	3	431,114
UniProt	1	3	4,405,573

*Transcript alignments and crossreference only.

Our complete schema mapping is available for download at `https://users.soe.ucsc.edu/~rhalpert/xr-benchmarks/`. Table 1 summarizes the data sources.

We represent the given part of UCSC's gene model with two tables, ComputedAlignments, which holds data about the transcripts themselves, and ComputedCrossref, which holds a cross-reference between UCSC "known gene IDs" (referred to here as "transcripts") and the closest corresponding external database transcript identifier (usually a RefSeq accession) and protein identifier (usually a UniProt or RefSeq accession). Our hand-written schema mapping specifies how these tables and the RefSeq, EntrezGene, and UniProt databases are used to populate the target schema. It also applies a key constraint to each target relation, per industry best-practice. Many of the key constraints are specified by the Genome Browser's schema, while some are not specified but are reasonable constraints that are in fact satisfied by the Genome Browser data.

The true Genome Browser's process computes a single coherent truth which may differ from that represented in the external databases. Our schema mapping, on the other hand, consolidates these sources into the target schema, and this gives rise to some inconsistency. The key constraints on the knownGene and kgXref tables prove critical in this regard: they enforce that each transcript have exactly one value for the exon count, and one gene symbol, respectively. Since values from both the UCSC gene model and from the other sources are used to populate the relevant attributes, this effectively gives rise to constraint violations when the UCSC gene model disagrees with RefSeq on the number of exons, and when RefSeq and EntrezGene collectively list more than one gene symbol. These two circumstances are expected to arise a small fraction of the time. The relevant parts of the schema mapping are depicted in Figure 2.

The knownIsoforms relation groups transcripts into clusters, where each cluster represents a gene. The Genome Browser computes this relation based on genomic coordinates [14]. Our schema mapping populates the knownIsoforms relation using a naive simplification of this approach: transcripts that share either an Entrez Gene ID or a gene symbol are made to reside in the same cluster. These two

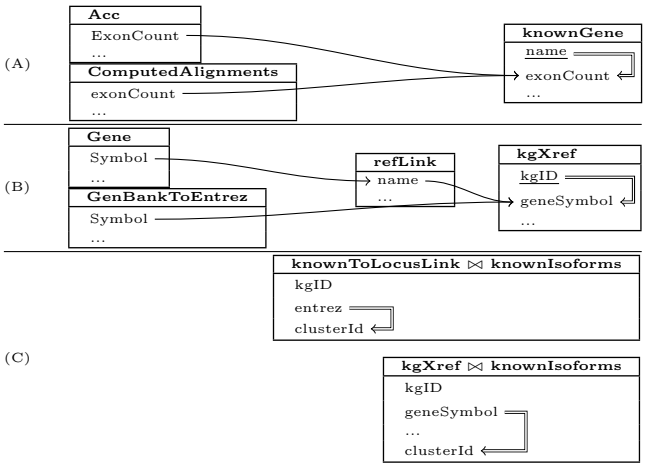

Figure 2: Critical parts of the schema mapping. Single arrows represent value propagation via tgds, and double arrows represent functional dependencies (egds). Keys (also egds) are underlined. (A) Competing values for the exon count. (B) Competing values for the gene symbol. (C) Clustering of transcripts according to Entrez Gene ID and gene symbol. The indicated egds give rise to equalities between nulls.

approaches are incomparable. Ours relies on existing gene symbol annotations from Entrez and UniProt, as well as crossreference annotations on UCSC transcripts, all of which have different levels of rigor and completeness. The knownIsoforms table is thus included in our schema mapping primarily to exercise the interaction of existentially quantified values with egds, which is a differentiating feature of weakly acyclic schema mappings versus other types of syntactic restrictions of GLAV+(GLAV, EGD) schema mappings (e.g., GAV+(GAV, EGD) or separable [6] schema mappings). This part of the schema mapping is depicted in Figure 2.

5.1 Benchmark Data and Queries

We wish to test the scalability of our implementation relative to two factors: the instance size and the proportion of the source tuples "involved" in egd violations (called "suspect" tuples), which we make precise with the notion of a *source repair envelope* defined in Section 6.2). To this end, we define a set of instances having specified sizes and ratios of suspect to total source tuples. In order to construct our test instances, we chase the unmodified source instances with the schema mapping and compute which source tuples are suspect. We call this the *raw result*. Then, each test instance is produced by selecting a randomized subset of the source instances with the desired size and ratio of suspect to total tuples in one particular table (ComputedCrossref), which we use as a rough proxy for the ratio for the entire source. Thus, for the largest size ("full"), the maximum ratio is the ratio found in the raw result (2.9%), and there is only one such instance. For the smaller instances, we are able to select enough suspect tuples to produce larger ratios. We use a randomized selection procedure to materialize a set of instances for each profile; six (at 3% suspect) for small and medium instances, three per ratio for large instances, and, necessarily, just one full instance (at 2.9% suspect). The characteristics of these instances are given in Table 2.

Table 2: Test instances have sizes small (S), medium (M), large (L), and full (F), and 0, 3, 9, or 20 percent of their transcripts suspect.

instance:	L0	L3	L9	L20
source tuples	321k	322k	316k	301k
total tuples	716k	724k	731k	748k
suspect transcripts	0%	3%	9%	20%
suspect tuples*	0%	2.0%	5.8%	13.4%

instance:	S3	M3	L3	F3
source tuples	3.5k	36k	322k	1,846k
total tuples	7.9k	77k	724k	5,354k
suspect transcripts	3%	3%	3%	2.9%
suspect tuples*	1.8%	1.8%	2.0%	3.4%

*includes source and target

Table 3 lists our query suite. Queries labeled "ep$N$" are adapted from the EQUIP query suite [16]: five of the 21 queries given there are applicable to our target schema. Additional queries labeled "xr$N$" are new queries created to exercise the critical parts of the schema mapping, including what is XR-Certain knowledge in the knownGene relation, and what pairs of transcripts reside in the same cluster in the knownIsoforms relation. In our experiments, we run the queries sequentially.

5.2 Monolithic Implementation and Results

Using the reduction given in Theorem 2, we have implemented a *monolithic* approach to XR-Certain query answering. Our monolithic implementation takes as input a GLAV+(WA-GLAV, EGD) schema mapping (encoded as text), an arbitrary source instance (via a JDBC connection string), and a union of conjunctive queries over the target schema (also text). The schema mapping is transformed into a GAV+(GAV, EGD) schema mapping using an optimized version of the reduction in Theorem 1, and the query is transformed into a new union of conjunctive queries using the same reduction. These transformations take an average of 18.7 seconds combined, and the resulting schema mapping is approximately seven times larger than the original (from 33 tgds and 26 egds to 339 tgds and 67 egds). We generate a separate disjunctive logic program for each query and instance, and run them using clingo 4.4.0, a solver from the Potsdam Answer Set Solving Collection[12] (Potassco).

All experiments are run on an 8-core Intel Core i7-2720QM CPU @2.20GHz with 16GB RAM, running Ubuntu 14.04 LTS (Linux 3.13.0 SMP x86_64). The plots of Figure 3 depict the runtime for these programs versus the percentage of suspect tuples and versus the instance size, respectively, with a line for each query. The latter is a log-log plot, since each instance size is an order of magnitude larger than the last.

These results illustrate a significant problem with this *monolithic* logic program approach: the cost of transforming the data from the source schema into the target schema is embedded in the execution cost of each individual query, which causes large instances to become unworkable even for simple queries. Additionally, the rapid increase in query runtimes as instance size increases, even for queries whose answers should be easy to compute, suggest that this implementation fails to take advantage of some exploitable structure in the instance and schema mapping. This is perhaps a symptom of the fact that the disjunctive logic program's rules are no simpler for areas of the source and target in-

Table 3: Query Suite, with approximate answer counts for the large-size instances.

Query	Answers
ep1() :- refLink(symbol, _, acc, protacc, _, _, _, _), kgXref(ucscid, _, spid, _, symbol, _, _, _, _, _)	1*
ep2(protacc) :- refLink(symbol, _, acc, protacc, _, _, _, _), kgXref(ucscid, _, spid, _, symbol, _, _, _, _, _)	6,000
ep3(protacc,spid) :- refLink(symbol, _, acc, protacc, _, _, _, _), kgXref(ucscid, _, spid, _, symbol, _, _, _, _, _)	12,000
ep15(symbol) :- kgXref(ucscid, _, _, _, symbol, refseq, _, _, _, _), refLink(_, product, refseq, _, _, _, entrez, _)	1,500
ep16(symbol,entrez) :- kgXref(ucscid, _, _, _, symbol, refseq, _, _, _, _), refLink(_, product, refseq, _, _, _, entrez, _)	1,500
xr1() :- knownGene(kgid, ch, sd, txs, txe, cs, ce, exc, exs, exe, pac, alignid)	1*
xr2(kgid) :- knownGene(kgid, ch, sd, txs, txe, cs, ce, exc, exs, exe, pac, alignid)	10,000
xr3(kgid, ch, sd, txs, txe, cs, ce, exc, exs, exe, pac, ai) :- knownGene(kgid, ch, sd, txs, txe, cs, ce, exc, exs, exe, pac, ai)	10,000**
xr4() :- knownIsoforms(cluster, transcript1), knownIsoforms(cluster, transcript2)	1*
xr5(transcript1) :- knownIsoforms(cluster, transcript1), knownIsoforms(cluster, transcript2)	10,000
xr6(transcript1, transcript2) :- knownIsoforms(cluster, transcript1), knownIsoforms(cluster, transcript2)	35,000

*boolean **projection-free

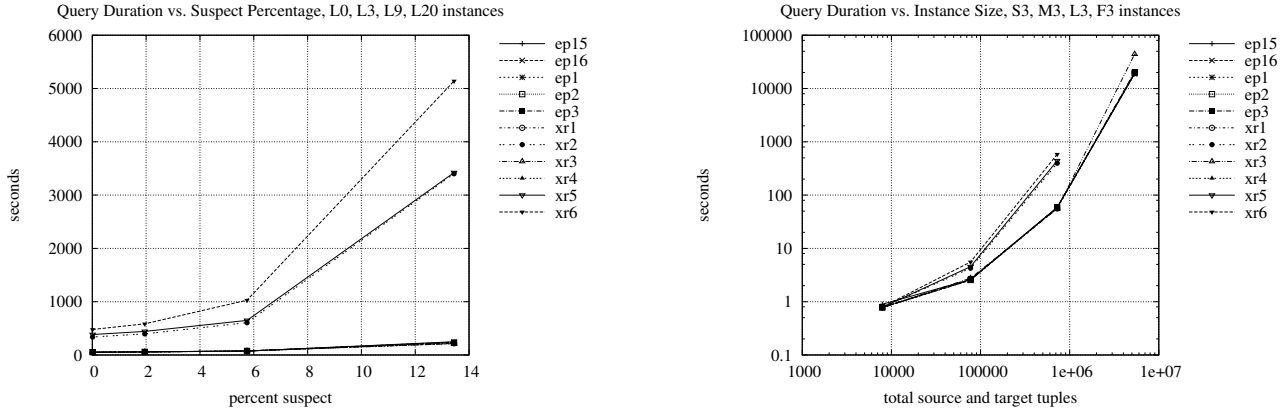

Figure 3: Performance of XR-Certain query answering using clingo.

stances that are unaffected by egd violations than for those that are affected. In the following section, we will develop techniques to identify and exploit such structure by grounding the egds.

6. ENHANCED APPROACH TO QUERY ANSWERING

Although the monolithic approach serves as a precise, straightforward specification of XR-Certain answers, it does not lend itself to fine-grained optimization. In this section, we present practical adaptations and optimizations to XR-Certain query answering that are motivated by our experimentation with the monolithic implementation, and that draw on techniques described in the literature for query answering over inconsistent databases, specifically, the notion of *repair envelopes* introduced by Eiter *et al.* [10]. We split query answering into two phases. The first, the "exchange phase", is a tractable-time query-independent preprocessing step, which enables the second, the "query phase", in which XR-Certain answers to a particular query are computed by solving a collection of small disjunctive logic programs. Although the problem at hand is coNP-complete, this new *segmentary approach* allows us to answer queries by solving many small hard problems rather than one large one. At the end of this section, we evaluate an implementation based on this enhanced approach.

6.1 Candidate Answers

The following definition of *candidate answers* effectively provides an upper bound on the set of XR-Certain answers of a query, in the sense that the latter is always a subset of the former.

Definition 2. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ be a GAV+(GAV, EGD) schema mapping.

1. We denote by $\mathcal{M}^{tgd}$ the schema mapping $(\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}}^{tgd})$ where Σ_{t}^{tgd} consists of the tgds from Σ_{t}. That is, all egds are dropped.
2. The *canonical quasi-solution* of a source instance I w.r.t. $\mathcal{M}$ is the canonical universal solution of I w.r.t. $\mathcal{M}^{tgd}$.
3. For every UCQ q over $\mathbf{T}$, the *candidate answers* to q w.r.t. I and $\mathcal{M}$ are $q(J)$, where J is the canonical quasi-solution of I w.r.t. $\mathcal{M}$.

PROPOSITION 1. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a* GAV+(GAV, EGD) *schema mapping. Let I be an $\mathbf{S}$-instance. Let J be the canonical quasi-solution of I w.r.t. $\mathcal{M}$.*

1. *For all canonical* XR-Solutions (I', J') *for I w.r.t. $\mathcal{M}$, it holds that $J' \subseteq J$.*
2. *For every UCQ q, we have that* XR-Certain$(q, I, \mathcal{M}) \subseteq q(J)$. *That is, every XR-Certain answer is indeed a candidate answer.*

PROOF. Let (I', J') be a canonical XR-Solution for I w.r.t. $\mathcal{M}$. Since $\mathcal{M}$ is GAV+(GAV, EGD) and J' is the canonical universal solution for I' w.r.t. $\mathcal{M}$, J' is simply the closure

of I' w.r.t. the tgds of $\mathcal{M}$. Therefore J' is also the canonical universal solution for I' w.r.t. $\mathcal{M}^{tgd}$. Finally, since the chase procedure is monotone [11] and $I' \subseteq I$, we have that $J' \subseteq J$.

The second item follows directly from the first, since UCQs are monotone queries. □

6.2 Source Repair Envelopes

It is often possible to exclude a large portion of the database from high-complexity computations. We will define a notion similar to a *repair envelope* from [10] but suited to the setting of data exchange.

Definition 3. Let $\mathcal{M}$ be a GLAV+(WA-GLAV, EGD) schema mapping and I a source instance. A subset E of I is a *source repair envelope* if $(I \setminus I') \subseteq E$ for all source repairs I'.

We will now see that we can restrict our attention within a source repair envelope when computing source repairs.

PROPOSITION 2. *Let $\mathcal{M}$ be a* GLAV+(WA-GLAV, EGD) *schema mapping, I a source instance, and $E \subseteq I$ a source repair envelope for I w.r.t. $\mathcal{M}$. Then $\{I' \mid I'$ is a source repair of I w.r.t. $\mathcal{M}\} = \{E' \cup (I \setminus E) \mid E'$ is a source repair of E w.r.t. $\mathcal{M}\}$.*

PROOF. Claim: $I' \cap E$ is a source repair of E.
Suppose for the sake of contradiction that $I' \cap E$ is not a source repair of E. Then either $I' \cap E$ has no solution (which cannot be the case due to monotonicity of the chase) or $I' \cap E$ is strictly contained in a source repair E'' of E. But then E'' must be contained in a source repair I'' of I. However, the definition of a source repair envelope tells us that I'' contains all of $E'' \cup (I \setminus E)$. Hence it contains I', so I' wasn't a source repair of I after all.

Claim: if E' is a source repair of E then $E' \cup (I \setminus E)$ is a source repair of I.
Indeed, E' must be contained in a source repair of I, and by the definition of a source repair envelope, that source repair of I must contain all of $(I \setminus E)$. Therefore it contains $E' \cup (I \setminus E)$. However, it cannot be a strict superset of $E' \cup (I \setminus E)$ because then E' could be extended to a larger source repair of E. □

There are many ways to calculate a source repair envelope (e.g., I is a trivial source repair envelope). Consider the *ideal source repair envelope*, given by $I - \bigcap\{I' \mid I'$ is a source repair for I w.r.t. $\mathcal{M}\}$. Equivalently, the ideal source repair envelope is the minimal source repair envelope for I w.r.t. $\mathcal{M}$. The next result tells us that computing this envelope is hard.

THEOREM 3. *Fix a schema mapping $\mathcal{M}$. Let the* intersection of source repairs membership problem *be the following decision problem: given a source instance I, and a fact f of I, is f contained in the intersection of all source repairs (that is, is $f \in \bigcap\{I' \mid I'$ is a source repair for I w.r.t. $\mathcal{M}\}$)? There is a* GAV+(GAV, EGD) *schema mapping for which this problem is* coNP*-hard.*

PROOF. The result is proven by reduction from the complement of 3-colorability. Let $G = (V, E)$ be a graph, with edge set $E = \{e_1, \ldots, e_n\}$. I_G is the source instance, with active domain $V \cup \{1, \ldots, n\}$, containing, for each edge $e_i = (a, b)$ the fact $E(a, b, i, i+1)$; for each vertex $a \in V$, the facts $C_r(a), C_g(a), C_b(a)$; and one additional fact, namely $F(n, 1)$.

$\mathcal{M}$ is the schema mapping consisting of the source-to-target tgds

- $E(x, y, u, v) \wedge C_z(x) \rightarrow E'(x, y)$ (for $z \in \{r, g, b\}$)
- $E(x, y, u, v) \wedge C_z(x) \rightarrow F'(u, v)$ (for $z \in \{r, g, b\}$)
- $P_z(x) \rightarrow P'_z(x)$ (for $z \in \{r, g, b\}$)
- $F(u, v) \rightarrow F'(u, v)$

and target constraints

- $E'(x, y) \wedge P'_z(x) \wedge P'_z(y) \wedge F'(u, v) \rightarrow u = v$ (for all $z \in \{r, g, b\}$)
- $F'(u, v) \wedge F'(v, w) \rightarrow F'(u, w)$
- $F'(u, u) \wedge F'(v, w) \rightarrow v = w$

Note that I has no solutions with respect to $\mathcal{M}$, regardless of whether G is 3-colorable. This is because a solution J of I has to contain a directed cycle of F'-edges (of length n), and F' must be transitive, which means that J would have to include facts of the form $F'(i, i)$, leading to an egd violation. Indeed, every source-repair of I must either (i) omit at least one of the E-facts, or (ii) omit all P_z-facts ($z \in \{r, g, b\}$) for some vertex or (iii) omit the $F(n, 1)$ fact. It is then not hard to see that G is 3-colorable if and only if some source repair omits $F(n, 1)$. Note that, if G is 3-colorable, then there is a source repair that retains all E-facts and that retains at least one P_z-fact for each vertex ($z \in \{r, g, b\}$). This source repair must then omit the $F(n, 1)$ fact. If, on the other hand, G is not 3-colorable, then every source repair has to satisfy (i) or (iii) and, consequently, will include $F(n, 1)$. □

The hardness established in Theorem 3 shows that computing the ideal source repair envelope is not helpful, given that the purpose of a source repair envelope is to help reduce the need for high-complexity computations. Our next result pertains to a source repair envelope that *can* be computed in PTIME.

First, we introduce the notion of *support sets* for a target fact. For an egd or GAV tgd σ and an instance I, we denote by $ground(\sigma, J)$ the set of all groundings of σ using values from the active domain of I (that is, quantifier-free formulas that can be obtained from σ by replacing universally quantified variables by values from the active domain of I). Note that, if J is a canonical quasi-solution for a source instance I w.r.t. a GAV+(GAV, EGD) schema mapping, then the active domain of J is already included in the active domain of I.

Definition 4. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ be a GAV+(GAV, EGD) schema mapping. Let I be an $\mathbf{S}$-instance with canonical quasi-solution J, and let $f \in J$ be a fact.

- A *support set* for f is a set of the form $\{f_1, \ldots, f_n\}$ where $(f_1 \wedge \cdots \wedge f_n \rightarrow f) \in \text{ground}(\Sigma_{\text{st}} \cup \Sigma_{\text{t}}, I)$, and $(I, J) \models f_1 \wedge \cdots \wedge f_n$. The set of all support sets of f is denoted by $\text{support\_sets}(f, I, \mathcal{M})$.
- The *support closure* for a set F of facts, denoted as $\text{support}^*(F, I, \mathcal{M})$, is the smallest set containing F such that whenever $g \in \text{support}^*(F, I, \mathcal{M})$, then all facts that belong to a support set of g belong to $\text{support}^*(F, I, \mathcal{M})$ as well.

Definition 5. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ be a GAV+(GAV, EGD) schema mapping. Let J be the canonical quasi-solution of I. Let

$$\text{violations}(I, \mathcal{M}) = \left\{ f \;\middle|\; \begin{array}{l} f \text{ occurs in the body of some} \\ \sigma \in \text{ground}(\Sigma_{\text{t}}, I) \text{ with } J \not\models \sigma \end{array} \right\}$$

We say that a source fact $f \in I$ is *suspect* (w.r.t. $\mathcal{M}$) if it belongs to support*(violations$(I, \mathcal{M}), I, \mathcal{M})$, and that it is *safe* otherwise. The set of suspect facts of I is denoted by I_{suspect} and the set of safe facts of I is denoted by I_{safe}.

The violation set is, intuitively, the set of facts that are directly involved in an egd violation, while the violation closure is, intuitively, the set of facts that are, possibly indirectly, involved in an egd violation. The notation I_{safe} and I_{suspect} assumes that it is clear from the context which schema mapping is being referred to.

PROPOSITION 3. *Let $\mathcal{M}$ be a* GAV+(GAV, EGD) *schema mapping and I a source instance. Then $I_{suspect}$ is a source repair envelope for I (w.r.t. $\mathcal{M}$). Moreover, $I_{suspect}$ can be computed in polynomial time (data complexity).*

To see that this proposition holds, suppose that some $f \in I$ is omitted by a source repair I' of I. Then $I' \cup \{f\}$ has no solution. It follows that f must be in the violation closure of I w.r.t. $\mathcal{M}$ (with the steps of the chase of $I' \cup \{f\}$ serving as proof of such). Therefore, f belongs to I_{suspect}.

The following example reminds us that I_{suspect} is not necessarily a *minimal* source repair envelope.

Example 1. Let $I = \{P(a,b), P(a,c), Q(b,c)\}$, and let $\mathcal{M} = (\{P,Q\}, \{P',Q'\}, \{P(x,y) \rightarrow P'(x,y), Q(x,y) \rightarrow Q'(x,y)\}, \{P'(x,y) \wedge P'(x,y') \rightarrow y = y', P'(x,y) \wedge P'(x,y') \wedge Q'(y,y') \rightarrow y = y'\})$. Then

$$I_{\text{suspect}} = \{P(a,b), P(a,c), Q(b,c)\}$$

However, the key constraint on P' forces every XR-Solution to have at most one of $P(a,b), P(a,c)$, so the second egd in Σ_{t} is satisfied without eliminating $Q(b,c)$. Indeed, the ideal source repair envelope for I is $\{P(a,b), P(a,c)\}$.

Nonetheless, I_{suspect} is a potentially useful source repair envelope: we vary the proportion of facts in I_{suspect} versus I_{safe} in our test instances in order to evaluate the importance of this measure.

We will now extend the notion of a source repair envelope to include target instances.

Definition 6. Let $\mathcal{M}$ be a GAV+(GAV, EGD) schema mapping, and I a source instance. Let J be the canonical quasi-solution of I. Two sets $E \subseteq I$ and $F \subseteq J$ of facts, together comprise an *exchange repair envelope* (E, F) if, for all canonical XR-Solutions (I', J') of I w.r.t. $\mathcal{M}$, we have that $(I \setminus I') \subseteq E$ and $(J \setminus J') \subseteq F$.

It follows from Theorem 3 that computing the minimal exchange repair envelope is hard. We will now see how to extend an existing source repair envelope to the target, in polynomial time.

Definition 7. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ be a GAV+(GAV, EGD) schema mapping, and I be an $\mathbf{S}$-instance. Let J be the canonical quasi-solution of I w.r.t. $\mathcal{M}$. Define the *influence* of a set of facts $E \subseteq I$, denoted influence$(E, I, \mathcal{M})$, as the smallest set containing E such that whenever $g \in$ influence$(E, I, \mathcal{M})$, every fact f that has a support set containing g also belongs to influence$(E, I, \mathcal{M})$.

PROPOSITION 4. *Let $\mathcal{M}$ be a* GAV+(GAV, EGD) *schema mapping and I a source instance. Let J be the canonical quasi-solution of I w.r.t. $\mathcal{M}$. Let E be a source repair envelope for I w.r.t. $\mathcal{M}$, and let $F = influence(E, I, \mathcal{M})$. Then (E, F) is an exchange repair envelope for I w.r.t. $\mathcal{M}$.*

In particular, $(I_{suspect}, J_{suspect})$ is an exchange repair envelope for I, where $J_{suspect} = influence(I_{suspect}, I, \mathcal{M})$.

PROOF. Let (I', J') be a canonical XR-Solution for I w.r.t. $\mathcal{M}$. Let t be a fact in $J \setminus J'$. Since t is not in (I', J'), there must be some fact f in support*$(\{t\}, I, \mathcal{M})$ in $I \setminus I'$. Therefore, f is contained in the source repair envelope E, so by definition we have that $t \in$ influence$(E, I, \mathcal{M})$. □

Fact 1. The following two statements hold, where F is an arbitrary set of target facts:

- The influence of the source restriction of the support closure of F contains the support closure of F; and
- A support closure of F and its influence are equal over their source restrictions.

In light of the above, we can refer to violation influences instead of violation closures whenever we wish to work with exchange repair envelopes rather than source repair envelopes. It is important to notice that a fact may have one support set which places it in a violation influence, but also another support set whose facts are not contained in any violation influence. Such facts lie in the difference between the violation influence and the ideal exchange repair envelope, but are nonetheless easy to identify.

6.3 Violation Clusters

Violation clusters are a concept that will help us further reduce the combinatorial complexity of computing XR-Certain answers. We start with a motivating example.

Example 2. Let $I = \{P_1(a,b), P_1(a,c), P_2(a,b), P_2(a,c), \ldots, P_n(a,b), P_n(a,c)\}$, and let

$$\begin{aligned} \mathcal{M} = (&\{P_1, \ldots, P_n\}, \{Q_1, \ldots, Q_n\}, \\ &\{P_1(x,y) \rightarrow Q_1(x,y), \ldots P_n(x,y) \rightarrow Q_n(x,y)\}, \\ &\{Q_1(x,y) \wedge Q_1(x,y') \rightarrow y = y', \ldots, \\ &\qquad Q_n(x,y) \wedge Q_n(x,y') \rightarrow y = y'\}) \end{aligned}$$

There are 2^n source repairs, which can be built by choosing one source atom from each of the n relations, in every possible combination. In this sense, the set of source repairs for this example is highly structured.

Now consider the query $q(x)$:- $Q_1(x,y)$. Every source repair of I w.r.t. $\mathcal{M}$ contains either $P_1(a,b)$ or $P_1(a,c)$, so we can conclude that $q(a) \in$ XR-Certain$(q, I, \mathcal{M})$ by considering just the two possibilities for the the P_1 relation. In so doing, we ignore the other $n-1$ relations and avoid having to consider 2^n source repairs.

In this section, we will generalize the above observation and demonstrate that it can be used to reduce the size of instances and schema mappings for which we must explore

all source repairs. To do so, we introduce a notion of *independence* that captures when particular egd violations are sufficiently isolated from each other to be processed separately.

To simplify the presentation, it will be convenient to consider schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ in which Σ_{t} may contain *grounded egds* (where universally quantified variables have been replaced by constants). This will allow us to more easily describe how an instance is carved up into segments that can be processed independently. Intuitively, each grounding of an egd describes one potential violation of that egd. The notions of solutions, universal solutions, source repairs, and exchange repair solutions all apply without modification to schema mappings containing grounded egds.

As a slight abuse of notation, when D is a set of target constraints, we will write $\mathcal{M} \cup D$ to denote the schema mapping obtained by adding the constraints in D to the target constraint set of a schema mapping $\mathcal{M}$.

Definition 8. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\text{t}})$ be a GAV+(GAV, EGD) schema mapping. Let I be an $\mathbf{S}$-instance. Let J be the canonical quasi-solution for I w.r.t. $\mathcal{M}$. Let $\sigma_1, \sigma_2 \in \text{ground}(\Sigma_{\text{t}}, I)$ be distinct grounded egds with $J \not\models \sigma_1$ and $J \not\models \sigma_2$. Let E_1 be the ideal source repair envelope for I w.r.t $(\mathcal{M}^{tgd} \cup \{\sigma_1\})$, and let E_2 be the ideal source repair envelope for I w.r.t $(\mathcal{M}^{tgd} \cup \{\sigma_2\})$. We say σ_1 and σ_2 are *pairwise-independent* if $\text{source\_repairs}(I, \mathcal{M}^{tgd} \cup \{\sigma_1, \sigma_2\}) = \{(I \setminus (E_1 \cup E_2)) \cup E_1' \cup E_2' \mid E_1' \in \text{source\_repairs}(I \cap E_1, \mathcal{M}^{tgd} \cup \{\sigma_1\})$ and $E_2' \in \text{source\_repairs}(I \cap E_2, \mathcal{M}^{tgd} \cup \{\sigma_2\})\}$. We say σ_1 and σ_2 are *pairwise-dependent* if they are not *pairwise-independent*.

Consider the graph of all egd violations in the quasi-solution and connect each pair of pairwise dependent egd violations by an edge. Each connected component of this graph is called a *violation cluster*. If σ_1 and σ_n reside in distinct violation clusters then we say σ_1 and σ_n are *independent*.

If a pair of violations is independent, by definition, their XR-Solutions can be processed separately, then suitably recombined. Note that the above definition of violation clusters does not provide us with a way to compute them efficiently, because the definition involves ideal source repair envelopes. The following proposition provides an approximation.

PROPOSITION 5. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a* GAV+(GAV, EGD) *schema mapping. Let I be an $\mathbf{S}$-instance. Let J be the canonical quasi-solution I w.r.t. $\mathcal{M}$. Let σ_1, σ_2 be distinct grounded egds in ground(Σ_t, I) such that $J \not\models \sigma_1$ and $J \not\models \sigma_2$. Let E_1, E_2 be ideal source repair envelopes for I w.r.t. $\mathcal{M} \cup \sigma_1$ and $\mathcal{M} \cup \sigma_2$, respectively. If E_1 and E_2 are disjoint, then σ_1 and σ_2 are pairwise-independent.*

PROOF. Suppose $E_1 \cap E_2 = \emptyset$, and let (I', J') be a canonical XR-Solution for I w.r.t. $\mathcal{M} \cup \{\sigma_1, \sigma_2\}$. Let $E_1' = E_1 \cap I'$ and let $E_2' = E_2 \cap I'$. By definition of a source repair envelope, $(I \setminus E_1) \cup E_1'$ is an XR-Solution for I w.r.t. $\mathcal{M} \cup \{\sigma_1\}$, and likewise $(I \setminus E_2) \cup E_2'$ is an XR-Solution for I w.r.t. $\mathcal{M} \cup \{\sigma_2\}$, and since E_1 and E_2 are disjoint, it is easy to see that E_1' is an XR-Solution for E_1 w.r.t. $\mathcal{M} \cup \{\sigma_1, \sigma_2\}$ and E_2' is an XR-Solution for E_2 w.r.t. $\mathcal{M} \cup \{\sigma_1, \sigma_2\}$. Finally, since GAV chase is monotone, we have that $(I \setminus E_1 \setminus E_2) \cup (E_1') \cup (E_2')$ has a solution w.r.t. $\mathcal{M} \cup \{\sigma_1, \sigma_2\}$, and there is no instance $I'' \subset I$ which strictly contains $(I \setminus E_1 \setminus E_2) \cup (E_1') \cup (E_2')$ and also has a solution w.r.t. $\mathcal{M} \cup \{\sigma_1, \sigma_2\}$. □

We can, in polynomial time, compute the support closure for each egd violation, then compute an overapproximation of the violation clusters based on the restriction to the source schema of those closures. The next proposition follows simply from the definition of a support closure, but provides some intuition and gives a shortcut for computing a source repair envelope for a violation cluster.

PROPOSITION 6. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a* GAV+(GAV, EGD) *schema mapping. Let I be an $\mathbf{S}$-instance. Let J be the canonical quasi-solution for I w.r.t. $\mathcal{M}$. Let $\sigma_1, \ldots, \sigma_n$ be a violation cluster (so each σ_i is a ground egd where $J \not\models \sigma_i$), with support closures $E_1, \ldots, E_n$. Then $E_1 \cup \ldots \cup E_n$ is the support closure of the facts in $\sigma_1, \ldots, \sigma_n$, and its $\mathbf{S}$-restriction is a source repair envelope for I w.r.t. $\mathcal{M}^{tgd} \cup \{\sigma_1, \ldots, \sigma_n\}$*

We have now seen how to compute a conservative approximation of the violation clusters for an instance w.r.t. a given schema mapping. Proposition 6 makes clear the fact that distinct violation clusters have disjoint source repair envelopes, and are therefore pairwise-independent themselves. Definition 8 tells us that we can thus compute the source repairs for an entire instance by computing separately the source repairs for the envelope of each violation cluster. We will now see how this supports query answering.

6.4 Answering Queries

We now show how to use the techniques introduced in the previous sections to compute XR-Certain answers for unions of conjunctive queries. In fact, without loss of generality, we can restrict attention to projection-free atomic queries: Let $q(\mathbf{x})$:- $\phi_1(\mathbf{x}, \mathbf{y}) \vee \ldots \vee \phi_n(\mathbf{x}, \mathbf{y})$ be a union of conjunctive queries with n clauses. Define $t_1, \ldots, t_n$ to be a set of new GAV tgds, where the head of each tgd is the head of q, and the body of each tgd t_k is $\phi_k(\mathbf{x}, \mathbf{y})$. If we extend a canonical quasi-solution J with the new relation symbol q, and chase with $t_1, \ldots, t_n$, the result will be $J \cup \{q(J)\}$. We will use the notation XR-Certain$(I, \mathcal{M} \cup \{t_1, \ldots, t_n\})$ to denote the set of facts in the intersection of all exchange repair solutions for I w.r.t. $\mathcal{M} \cup \{t_1, \ldots, t_n\}$, and the term *candidate facts* to refer to the tuples of any relation in $\mathbf{T} \cup \text{q}$.

Consider the definition of *support sets* in Section 6.2, and suppose f is a candidate fact. By definition, f is XR-Certain if it is contained in every exchange repair solution. Since exchange repair solutions satisfy the constraints of the schema mapping, it is easy to see that f is XR-Certain if it has at least one support set in every XR-Solution, or equivalently, in every canonical XR-Solution.

Proposition 6 naturally extends to exchange repair envelopes. Thus we define a *violation cluster influence* to refer to the union of the influences of the violations in a cluster.

Example 3. This example illustrates that a candidate fact f may belong to the influences of multiple distinct violation clusters. Let $I = \{P(a_1, a_2), P(a_1, a_3), Q(a_1, a_2), Q(a_1, a_3)\}$, and let $\mathcal{M} = (\{P, Q\}, \{R, S, T\}, \{P(x, y) \rightarrow R(x, y), Q(x, y) \rightarrow S(x, y)\}, \{R(x, y) \wedge S(x, z) \rightarrow T(x, y, z), R(x, y) \wedge R(x, y') \rightarrow$

$y = y', S(x,y), S(x,y') \rightarrow y = y'\}$). Then the violation cluster influence for $\{R(a_1,a_2), R(a_1,a_3)\}$ is

$$\left\{\begin{matrix} P(a_1,a_2), P(a_1,a_3), R(a_1,a_2), R(a_1,a_3), \\ T(a_1,a_2,a_3), T(a_1,a_3,a_2), T(a_1,a_2,a_2), T(a_1,a_3,a_3) \end{matrix}\right\}$$

and the violation cluster influence for $S(a_1,a_2), S(a_1,a_3)$ is

$$\left\{\begin{matrix} Q(a_1,a_2), Q(a_1,a_3), S(a_1,a_2), S(a_1,a_3), \\ T(a_1,a_2,a_3), T(a_1,a_3,a_2), T(a_1,a_2,a_2), T(a_1,a_3,a_3) \end{matrix}\right\}$$

which are disjoint in their restriction to the source schema, yet both contain the target facts $T(a_1,a_2,a_3)$, $T(a_1,a_3,a_2)$, $T(a_1,a_2,a_2)$, and $T(a_1,a_3,a_3)$.

This example illustrates how distinct target violations with non-overlapping source repair envelopes may jointly affect target tuples; we cannot determine the status of tuples in T without considering violations of both key constraints.

Suppose f is a candidate fact in T. Each support set for f may be contained in only certain combinations of XR-Solutions from the violation cluster influences containing f. So, to determine if f has at least one support set in every XR-Solution w.r.t. the broader schema mapping, it is necessary to consider **all** combinations of XR-Solutions from the violation cluster influences containing f. We call the set of violation clusters whose influences contain f the *signature* of f, denoted signature(f). Recall that I_{safe} denotes the set of source facts that are *safe*, that is, not *suspect* (Definition 5). In the following, let J_{safe} denote $chase(I_{\text{safe}}, \mathcal{M})$.

THEOREM 4. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a* GAV+(GAV, EGD) *schema mapping. Let I be an $\mathbf{S}$-instance. Let J be the canonical quasi-solution for I w.r.t. $\mathcal{M}$. Let f be a candidate fact in J. Let $I_{f\text{-focus}}$ and $J_{f\text{-focus}}$ be, respectively, the source and target parts of $\bigcup\{V \mid V$ the influence for a violation cluster in* signature(f)$\}$. *Then $f \in$ XR-Certain$(I, \mathcal{M}) \leftrightarrow f \in \bigcap\{chase(J'_{f\text{-focus}} \cup J_{safe}, \Sigma_t) \mid J'_{f\text{-focus}} \in$ XR-Solution$(I_{f\text{-focus}}, \mathcal{M})\}$.*

PROOF SKETCH. By definition, the violations in the clusters in signature(f) are contained in $J_{f\text{-focus}}$. Thus $(I_{f\text{-focus}}, J_{f\text{-focus}})$ is an exchange repair envelope for I w.r.t. $\mathcal{M}^{\text{tgd}}$ augmented with those violations. Furthermore, all of the violations in $J_{f\text{-focus}}$ are pairwise independent of all of the violations in $J \setminus J_{f\text{-focus}}$, from which we conclude that every fact in every support set for f is contained in $(I_{f\text{-focus}}, J_{f\text{-focus}})$ or in $(I_{\text{safe}}, J_{\text{safe}})$. □

This result gives us the following algorithm for computing XR-Certain for an instance I and schema mapping $\mathcal{M}$, using a (hopefully large) J_{safe} combined with, for each signature, a (hopefully small) J_{focus}. For the exchange phase: Chase I with $\mathcal{M}^{tgd}$, compute the violation set of I w.r.t. $\mathcal{M}$, and compute the support closure of each violation. Then mark source facts safe if they do not reside in any violation closure, chase I_{safe} with $\mathcal{M}^{tgd}$, and mark every resulting fact safe. Lastly, compute violation clusters, and the influence of each cluster. For the query phase: Compute the candidate facts, marking safe those with support sets in J_{safe}. Next, compute the signature of each unmarked candidate fact. Finally, generate and solve a grounded disjunctive logic program to compute XR-Certain w.r.t. $\mathcal{M}$ for $(I_{\text{focus}}, J_{\text{focus}})$ for each signature. This program is the restriction to $(I_{\text{focus}}, J_{\text{focus}})$ of the grounding of the program from Theorem 2. Facts in $(I_{\text{focus}}, J_{\text{focus}}) \cap (I_{\text{safe}}, J_{\text{safe}})$ may be represented by the value TRUE in the program.

6.5 Segmentary Implementation and Results

Using the techniques developed in this section, we have implemented a *segmentary* approach to XR-Certain query answering using Java 1.7.0_80, MySQL 5.5.42, and clingo 4.4.0. Our segmentary implementation takes as input a GLAV+(WA-GLAV, EGD) schema mapping (encoded as text), an arbitrary source instance (via a JDBC connection string), and a union of conjunctive queries over the target schema (also text). As with the monolithic implementation, the schema mapping is transformed into a GAV+(GAV, EGD) schema mapping. Additionally, the query is transformed into an atomic query using the reduction described at the beginning of Section 6.4.

Using the above algorithm, query answering is done in two phases: the exchange phase, and the query phase. The exchange phase materializes the target instance in MySQL using a chase procedure written in Java. The detailed implementation of the chase procedure is immaterial: here, we use a semi-naïve chase. The exchange phase next computes violation cluster influences, fact signatures, and the "safe" part of the source and target instances (also in MySQL, run from Java). The query phase appends candidate answers to the target instance, marks "safe" candidates, and generated disjunctive logic programs as explained in Section 6.4, which are then solved using clingo. The results obtained from clingo are used to mark each candidate either "accepted" or "rejected". The "safe" candidates and "accepted" candidates together comprise the XR-Certain query answers.

Table 4: Duration of the exchange phase, in seconds.

instance	L0	L3	L9	L20
duration	150.7	196.7	235.7	297.3
instance	S3	M3	L3	F3
duration	36.5	50.8	196.7	2229.7

Table 4 gives the runtime of the exchange phase for each instance. Notice that for large instances, the exchange phase compares very favorably against the per-query runtime of the monolithic approach described in Section 5.2. The plots in Figure 4 give the performance of each query as we scale the rate of violations and the instance size, respectively, with the latter on a log-log scale. These results improve considerably on the monolithic approach: the query phase runtimes are between ten times and one thousand times faster for large and full instances.

7. CONCLUDING REMARKS

We have implemented XR-Certain query answering and evaluated it on real data using a benchmark that mimicks the UCSC Genome Browser data import process. Our experiments suggest that using the reduction from XR-Certain to disjunctive logic programming to create a monolithic logic program is not a viable approach using today's best-of-breed solvers. However, by efficiently computing a suitable overapproximation, our segmentary approach computes query answers ten to one thousand times faster for larger instances, and exhibits promising scalability with respect to both instance size and the rate of target constraint violations.

We note that significant progress has recently been made toward broadly applicable, reproducible schema mapping benchmarks, in particular iBench [3]. We intend as fu-

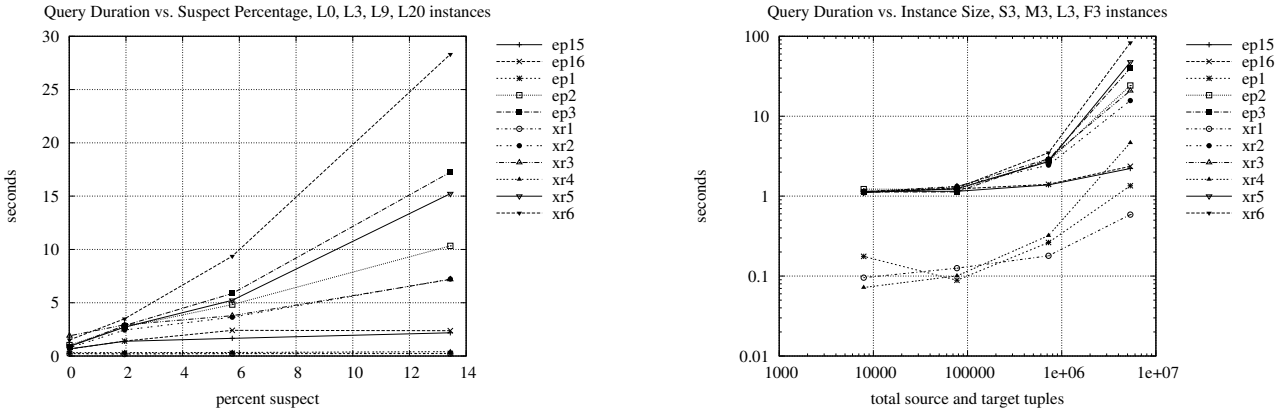

Figure 4: Performance of XR-Certain query answering using MySQL along with clingo.

ture work to further evaluate our segmentary implementation on such benchmarks. Nonetheless, our success with our Genome Browser benchmark serves as evidence that XR-Certain query answering may be efficiently computable in practice for realistic applications.

8. ACKNOWLEDGMENTS

The research of all authors was partially supported by NSF Grant IIS-1217869. Kolaitis' research was also supported by the project "Handling Uncertainty in Data Intensive Applications" under the program THALES.

9. REFERENCES

[1] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Relational and XML Data Exchange*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[2] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In V. Vianu and C. H. Papadimitriou, editors, *PODS*, pages 68–79. ACM Press, 1999.

[3] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller. The iBench integration metadata generator. *PVLDB*, 9(3):108–119, 2015.

[4] L. E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[5] M. Bienvenu and M. Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web. Web Logic Rules - 11th Int. Summer School*, pages 218–307, 2015.

[6] A. Calì, M. Console, and R. Frosini. Deep separability of ontological constraints. *CoRR*, abs/1312.5914, 2013.

[7] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In F. Neven, C. Beeri, and T. Milo, editors, *PODS*, pages 260–271. ACM, 2003.

[8] B. ten Cate, R. L. Halpert, and P. G. Kolaitis. Exchange-repairs: Managing inconsistency in data exchange. In *RR*, volume 8741 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2014.

[9] B. ten Cate, R. L. Halpert, and P. G. Kolaitis. Exchange-repairs: Managing inconsistency in data exchange, September 2015. arXiv `http://arxiv.org/abs/1509.06390`, 29 pages; to appear in the *Journal of Data Semantics*.

[10] T. Eiter, M. Fink, G. Greco, and D. Lembo. Repair localization for query answering from inconsistent databases. *ACM Trans. Database Syst.*, 33(2), 2008.

[11] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[12] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.

[13] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.

[14] F. Hsu, W. J. Kent, H. Clawson, R. M. Kuhn, M. Diekhans, and D. Haussler. The UCSC known genes. *Bioinformatics*, 22(9):1036–1046, 2006.

[15] T. Janhunen and E. Oikarinen. Capturing parallel circumscription with disjunctive logic programs. In J. J. Alferes and J. A. Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 134–146. Springer, 2004.

[16] P. G. Kolaitis, E. Pema, and W.-C. Tan. Efficient querying of inconsistent databases with binary integer programming. *PVLDB*, 6(6):397–408, 2013.

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

[18] M. C. Marileo and L. E. Bertossi. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data Knowl. Eng.*, 69(6):545–572, 2010.

[19] National Center for Biotechnology Information. NCBI ToolBox, 2001. `http://www.ncbi.nlm.nih.gov/IEB/ToolBox/`.

[20] National Center for Biotechnology Information. Entrez Programming Utilities, 2013. `http://www.ncbi.nlm.nih.gov/books/NBK25501/`.