On the Complexity of Deriving Schema Mappings from Database Instances

Pierre Senellart*
INRIA Saclay – Île-de-France
& Université Paris-Sud
Orsay, France
pierre@senellart.com

Georg Gottlob†
Computing Laboratory
& Man Institute of Quantitative Finance
University of Oxford, United Kingdom
Georg.Gottlob@comlab.ox.ac.uk

ABSTRACT

We introduce a theoretical framework for discovering relationships between two database instances over distinct and unknown schemata. This framework is grounded in the context of *data exchange*. We formalize the problem of understanding the relationship between two instances as that of obtaining a schema mapping so that a *minimum repair* of this mapping provides a perfect description of the target instance given the source instance. We show that this definition yields "intuitive" results when applied on database instances derived from each other by basic operations. We study the complexity of decision problems related to this optimality notion in the context of different logical languages and show that, even in very restricted cases, the problem is of high complexity.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; H.2.5 [**Database Management**]: Heterogeneous Databases

General Terms

Languages, Theory

Keywords

Schema mapping, instance, complexity, match, data exchange

1. INTRODUCTION

Main Problem Addressed. This paper deals with the automatic discovery of relational schema mappings based on existing data. In particular, we deal with the following main problem, and with closely related questions. Given a relational schema $\mathbf{S}$, called the *source schema*, and a differently structured *target schema* $\mathbf{T}$, and given an instance I of $\mathbf{S}$, and an instance J of $\mathbf{T}$, where we assume that J consists of an adequate "translation" of I to fit the target schema $\mathbf{T}$, find an optimal translation function, i.e., a schema mapping that maps instances of $\mathbf{S}$ to instances of $\mathbf{T}$, taking I to J. This main problem actually consists of two important subproblems: (i) determining an appropriate formal framework in which schema mappings can be expressed and that allow one to numerically assess the quality of a schema mapping, and (ii) understanding the computational fundamentals of the automatic synthesis of schema mappings in that framework, in particular, complexity issues and algorithms. We do not provide a direct algorithm for deriving schema mappings from database instances, but we discuss a theoretical framework and complexity analysis that can be a first step toward it.

Importance of the Problem. Schema mapping discovery has been recognized as an issue of central relevance in various application and research contexts, for example, in the area of data exchange [7, 15], data integration [18], metadata management [3], and data extraction from the hidden Web, in particular, automated wrapper generation.

Schemata and dependencies, in a data exchange context, form *metadata* that need to be managed in a systematic and formal way. Bernstein argues in [3] for the definition, in such a setting, of *operators* on this metadata. Thus, [9] and [8] respectively propose ways to define the composition and inverse operators on schema mappings. Another operator of importance, which is actually closely related to the research proposed in the present paper, is the *match* operator [3]: given two schemata and instances of these schemata, how to derive an appropriate set of dependencies between these schemata. More precisely, given two relational databases schemata $\mathbf{S}$ and $\mathbf{T}$ and instances I and J of these schemata, the problem is to find a *schema mapping*, that is, a finite set Σ of formulas in a given language $\mathcal{L}$, such that $(I, J) \models \Sigma$, or such that (I, J) *approximates* in some sense a model of Σ. This problem is related to the techniques used for *automatic schema matching*. Current methods of automated schema mapping generation, such as those described in [20], heavily rely on semantic meta-information about the schemata (names of concepts and relations, concrete data types, etc.). However, in many practical contexts such semantic meta-information is either not available or would require too much or too expensive human interaction. In those cases, the mere values of I and J

*Partially supported by the ANR project WebContent.
†Georg Gottlob's work was supported by EPSRC grant EP/E010865/1 "Schema Mappings and Automated Services for Data Integration and Exchange". Gottlob also gratefully acknowledges a Royal Society Wolfson Research Merit Award, which allowed him to host Pierre Senellart.

PODS'08, June 9–12, 2008, Vancouver, BC, Canada.

constitutes the only information from which a mapping ought to be constructed. This important case, which has been barely studied, is addressed in the present paper.

In automated wrapper generation (e.g., from Web sources), the absence of suitable meta-information is a similar problem. Let us take a concrete example, namely that of extracting information from research publications databases on the Web. Consider for instance the list of publications by J. D. Ullman provided by DBLP[1] and Google Scholar[2]. A structural information extraction wrapper such as RoadRunner [5] can be applied on both pages (or set of pages obtained by following the *Next* links) to obtain relational data from these pages, *without any metadata*. The set of papers presented by both sources is not exactly the same, their organization is different (for instance, grouping by dates in DBLP), some data is present in some source and not in the other (page numbers in DBLP, direct links to electronic versions of articles in Google Scholar), but both sources essentially present information about the same data. Using the mutual redundancy of these sources to detect the most appropriate schema mapping between them would enable us to wrap from one format of output to another. If the structure of one source is known, then this schema mapping would give us the core structure of the other one, in a fully automatic way.

Results. After stating in Section 2 some relevant definitions, in Section 3 of this paper, we present a novel formal framework for defining and studying the automatic discovery of schema mappings. In this framework, schema mappings are—not surprisingly—expressed as source-to-target tuple generating dependencies (tgds). It is well known that tgds are suited for this task. However, we also introduce a cost function, that tells us how well a tgd does its job of translating the given source instance I into the given target instance J. This cost function takes into account (i) the size of the tgd, (ii) the number of *repairs* that have to be applied to the tgd in order for it to be valid and to perfectly explain all facts of J.

Of course, defining a cost function may be seen as a somewhat arbitrary choice. However, in Section 4, we give formal evidence of the appropriateness of the cost function, showing that it enjoys nice properties when I and J are derived from each other with elementary relational operations.

We study in Section 5 the computational complexity of the relevant problems. In particular, we show that computing the cost of a schema mapping lies at the third level of the polynomial hierarchy, while either fullness or acyclicity conditions reduce this complexity by one level. The problem is thus **NP**-complete for full acyclic tgds, and it remains **NP**-complete even in a very simple case where there is only one relation of arity 3 in the source schema, and one relation of arity 1 in the target schema. To see that, we use a lemma on the complexity of the Vertex-Cover problem in r-partite r-uniform hypergraphs, which is interesting in itself. Due to space constraints, most proofs are omitted.

We finally discuss in Section 6 an extension and variants of this approach. We show in particular how the cost definition can be extended to a schema mapping expressed as an arbitrary first-order formula and discuss the complexity of the relevant problems. We also examine other choices for the cost function, which may seem simpler at first and closer to the existing notion of *repair* of a database in the literature [1], but which are not appropriate to our context since they do not share the same "niceness" properties established in Section 4.

Related Work. We are not aware of any work with the same focus on a theoretical and systematic analysis of the complexity of deriving a schema mapping, although, in spirit, the problem that we deal with here is similar to the one that inductive logic programming (ILP) [17] aims to solve. An approach to the complexity of ILP is presented in [12]; the main differences with the work discussed here is the use of negative examples, the existence of a background knowledge, and the restriction to Horn clauses instead of arbitrary tgds.

The notion of *repair* appears in the context of inconsistent databases [1] (with respect to some integrity constraint). In this work, consistent query answers are defined as the common answers to a query on minimal repairs of the database. Repairs use the addition or deletion of tuples to the database, something close to what is discussed in Section 6 and that we show inappropriate to our context. Besides, the focus is different: Arenas et al. suppose the integrity constraint fixed, while we are looking for an optimal schema mapping without any *a priori*.

2. PRELIMINARIES

We assume some countably infinite sets $\mathcal{C}$ of constants (denoted a, b, 0, 1, etc.) and $\mathcal{V}$ of variables (denoted x, y, z, etc.). We use the notation $\mathbf{x}$ to represent a vector of variables $x_1 \dots x_n$. Constants appearing in formulas are here identified, as usual, with the domain elements they are interpreted by.

A (relational) schema is a finite set of pairs (R, n) where R is a relation name and $n \geqslant 1$ the arity of the relation. An instance I of a relational schema $\mathbf{S}$ consists, for every $(R, n) \in \mathbf{S}$, of a *finite* relation over $\mathcal{C}^n$. We occasionally denote R^I the interpretation of the relation name R in the instance I (if $|\mathbf{S}| = 1$, we shall make the confusion $R^I{=}I$). In the following, we assume that the schemata are implicitly given whenever we are given an instance.

A *language* $\mathcal{L}$ is a subset of the set of formulas of first-order logic with equality and constants, and without function symbols (with its usual semantics). Given a language $\mathcal{L}$, a *schema mapping* in $\mathcal{L}$ is a finite set of formulas in $\mathcal{L}$. We are particularly interested in the following languages, given instances I, J with schemata $\mathbf{S}$, $\mathbf{T}$:

Relational calculus. $\mathcal{L}_{\text{rc}}$ is the set of first-order formulas without constants, with relations symbols in $\mathbf{S} \cup \mathbf{T}$.

Source-to-target tuple-generating dependencies. $\mathcal{L}_{\text{tgd}} \subset \mathcal{L}_{\text{rc}}$ is the set of formulas of the form

$$\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$$

where: (i) $\varphi(\mathbf{x})$ is a (possibly empty) conjunction of positive relation atoms, with relation symbols in $\mathbf{S}$; (ii) $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of positive relation atoms, with relation symbols in $\mathbf{T}$; (iii) all variables of $\mathbf{x}$ appear in $\varphi(\mathbf{x})$.

Acyclic tgds. $\mathcal{L}_{\text{acyc}} \subset \mathcal{L}_{\text{tgd}}$ is the set of tgds such that the hypergraph of the relations on the left hand-side is acyclic [2], as well as the hypergraph of the relations on the right hand-side, considering only existentially quantified variables. More precisely, let $\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$

[1] http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/u/Ullman:Jeffrey_D=.html

[2] http://scholar.google.com/scholar?q=author%3A%22jd+ullman%22

be a tgd, and let (N, E) (respectively, (N', E')) be the hypergraph whose vertices are the variables of $\mathbf{x}$ (respectively, $\mathbf{y}$) and whose edges are the relation atoms of $\varphi(\mathbf{x})$ (respectively, the relation atoms of $\psi(\mathbf{x}, \mathbf{y})$ where at least one variable of $\mathbf{y}$ appears). The tgd is said to be acyclic[3] if there are two forests F and F' (called the *join forests*) with each hyperedge of E (respectively, of E') a node of F (respectively, of F'), such that for all nodes $n \in N$ (respectively, $n \in N'$), the subgraph of F (respectively, of F') induced by the edges of E (respectively, of E') that contain n is connected. Other equivalent definitions of acyclic hypergraphs are given in [2].

Full tgds. $\mathcal{L}_{\text{full}} \subset \mathcal{L}_{\text{tgd}}$ is the set of tgds without an existential qualifier on the right-hand side, that is, of the form

$$\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x}).$$

Acyclic full tgds. $\mathcal{L}_{\text{facyc}} = \mathcal{L}_{\text{acyc}} \cap \mathcal{L}_{\text{full}}$ is the set of full tgds such that the hypergraph of the relations on the left hand-side is acyclic.

We focus here on source-to-target tuple-generating dependencies (either arbitrary or with one of the restrictions mentioned above). Arbitrary tgds (and, in a lesser way, full tgds) have been at the basis of most works[4] in the data exchange setting [7, 15]. As we shall see in Section 5, acyclic full tgds have nice complexity results. We show in Section 6 how this work can be extended to arbitrary formulas of the relational calculus.

3. COST AND OPTIMALITY OF A TGD

We first introduce the two basic notions of *validity* and *explanation* that are at the basis of our framework.

Definition 1. A schema mapping Σ is *valid* with respect to a pair of instances (I, J) if $(I, J) \models \Sigma$.

Definition 2. A (ground) fact in a schema $\mathbf{S}$ is a tuple $R(c_1 \dots c_n)$ where $c_1 \dots c_n \in \mathcal{C}$ and R is a relation of $\mathbf{S}$ with arity n.

A schema mapping Σ *explains* a ground fact f in the target schema with respect to a source instance I if, for all instances K of the target schema such that $(I, K) \models \Sigma$, $f \in K$.

A schema mapping *fully explains* a target instance J with respect to a source instance I if it explains all facts of J with respect to I.

We have quite an asymmetric point of view about the pair of instances here; we do not require a full explanation of I by the facts of J, for instance. This asymmetry is quite common in the context of data exchange. For source-to-target tgds, note that Σ fully explains J with respect to I if and only if J is included in the result of the chase [7] of I by Σ.

Example 3. Let us consider the following database instances I and J, on schemata $\{(R, 1)\}$ and $\{(R', 2)\}$.

R
a
b
c
d

R'	
a	a
b	b
c	a
d	d
g	h

We can imagine a number of schema mappings that more or less express the relation between I and J:

$$\Sigma_0 = \varnothing$$
$$\Sigma_1 = \left\{\forall x\ R(x) \rightarrow R'(x, x)\right\}$$
$$\Sigma_2 = \left\{\forall x\ R(x) \rightarrow \exists y\ R'(x, y)\right\}$$
$$\Sigma_3 = \left\{\forall x \forall y\ R(x) \wedge R(y) \rightarrow R'(x, y)\right\}$$
$$\Sigma_4 = \left\{\exists x \exists y\ R'(x, y)\right\}$$

Actually, any combination of these schema mappings may also be of interest.

Σ_0 and Σ_4 seem pretty poor, here, as they fail to explain any facts of J, while there seems to be a definite relation (albeit with some noise) between I and J. Σ_3 explains most of the facts of J, but is far from being valid, since it also explains a large number of incorrect facts such as $R'(a, b)$ or $R'(b, d)$. Σ_1 and Σ_2 are more interesting. Σ_1 explains 3 facts of J, but also incorrectly predicts $R'(c, c)$. Σ_2 fails to explain any facts of J, but explain most of them at least partially, in the sense that they are explained by a fact with an existentially quantified variable (a *skolem*); in addition, it is valid with respect to (I, J). Neither Σ_1 or Σ_2 explains the last fact of J.

As there seems to be some noise in the operation that produced J from I, it is hard to say with certainty which schema mapping is optimal here, in the sense that it reflects most closely the relation between I and J. At any rate, however, Σ_1 and Σ_2 seem far better candidates than the other ones.

To define in a formal way our notion of optimality of a schema mapping, the basic idea is to get the simultaneous optimal for all three factors of interest (validity, explanation of the target instance, conciseness) by minimizing the size of: the original formula, plus all the local corrections that have to be done for the formula to be valid and to fully explain the target instance. This is close in spirit to the notion of Kolmogorov complexity and Kolmogorov optimal [19] (though we do not consider a Turing-complete language for expressing either the formula or the corrections, but much more limited languages).

Definition 4. Given a schema mapping of tgds $\Sigma \subset \mathcal{L}_{\text{tgd}}$ and a pair of instances (I, J), we define the set of *repairs* of Σ with respect to (I, J), denoted $\mathsf{repairs}_{(I,J)}(\Sigma)$, as a set of finite sets of formulas, such that $\Sigma' \in \mathsf{repairs}_{(I,J)}(\Sigma)$ if it can be obtained from Σ by a finite sequence of the following operations:

- Adding to the left-hand side of a tgd θ of Σ, with θ of the form $\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$, a conjunction $\tau(\mathbf{x})$ of the form: $\bigwedge_i x_i \alpha_i c_i$ where α_i are either $=$ or $\neq$, x_i are variables from $\mathbf{x}$ and c_i are constants.
- Adding to the right-hand side of a tgd θ of Σ, with θ as above, a formula $\tau'(\mathbf{x}, \mathbf{y})$ of the form:

$$\bigwedge_i \left(\left(\bigwedge_j x_{ij} = c'_{ij} \right) \rightarrow y_i = c_i \right)$$

[3]Note that this notion of acyclicity is not related to the notion of weakly acyclic set of tgds that has been much studied in the context of data exchange [7, 15].

[4]The other important class of dependencies, namely equality generating dependencies, is less appropriate to this context.

where x_{ij} are variables from $\mathbf{x}$, y_i variables from $\mathbf{y}$, and c'_{ij} and c_i constants.

- Adding to Σ a ground fact $R(c_1 \dots c_n)$ where R is a relation of the target schema of arity n, and $c_1 \dots c_n$ are constants.

The language of repairs $\mathcal{L}^*$ of a language $\mathcal{L}$ is the language consisting of all formulas which can be obtained from formulas of $\mathcal{L}$ with these operations (along with all ground facts over the target schema).

In a repair of a tgd $\forall \mathbf{x}\ \varphi(\mathbf{x}) \wedge \tau(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}) \wedge \tau'(\mathbf{x}, \mathbf{y})$, the term $\tau(\mathbf{x})$ is responsible for correcting cases when the tgd is not valid, by adding additional constraints on the universal quantifier, whereas $\tau'(\mathbf{x}, \mathbf{y})$ specifies the right-hand side of J, by giving the explicit value of each existentially quantified variable, in terms of the universally quantified variables.

An interesting property of repairs is that they are reversible: Because all operations add constants to a language where constants do not exist, it is possible to compute (in linear time) the original schema mapping from a repair. Indeed, constants are only used for repairing formulas; in other words, we consider that the relations that we need to find between the source and target instances are to be expressed with constant-free formulas, in order to abstract them as much as possible. Clearly, this is a simplifying assumption that could be lifted in future works. Note that this extension is not straightforward, however: It is not clear how to distinguish between constants which are rightfully part of the optimal schema mapping description and constants which are just used to correct some noise or missing data.

The notion of size of a schema mapping is easily defined as follows; we could also use a more classical definition in function of the number of symbols of a formula, without much difference in the theory.

Definition 5. The *size* of a first-order formula $\varphi \in \mathcal{L}^*$, denoted $\mathsf{size}(\varphi)$ is computed as follows:

- The size of φ is the number of occurrences of variables and constants in φ (we stress that each variable and constant is counted as many times as it occurs in φ); occurrences of variables as arguments of quantifiers do not count.
- If φ is a ground fact $R(c_1 \dots c_n)$, then the size of φ is computed as if φ were the formula

  $$\exists x_1 \dots \exists x_n\ R(x_1 \dots x_n) \wedge x_1 = c_1 \wedge \dots \wedge x_n = c_n.$$

  Therefore, $\mathsf{size}(\varphi) = 3n$.

The size of a schema mapping is the sum of the size of its elements.

The refinement on ground facts is performed so that such facts are not "too cheap": the cost of $R(c_1 \dots c_n)$ is the same as that of the corresponding repair of $\exists x_1 \dots \exists x_n\ R(x_1 \dots x_n)$, as will be illustrated in Example 8. This is not a major assumption, however, and it does not impact the results of this paper in a significant way. We are now ready to define the cost of a schema mapping, in terms of the size of its repairs:

Definition 6. The *cost* of a schema mapping Σ, with respect to a pair of instances (I, J), is defined by:

$$\mathsf{cost}_{(I,J)}(\Sigma) = \min_{\substack{\Sigma' \in \mathsf{repairs}_{(I,J)}(\Sigma) \\ \Sigma' \text{ valid and fully explains } J}} \mathsf{size}(\Sigma').$$

Note that $\mathsf{cost}_{(I,J)}(\Sigma)$ may not be defined if the minimizing set is empty.

A schema mapping $\Sigma \subset \mathcal{L}$ is *optimal* in the language $\mathcal{L}$, with respect to a pair of instances (I, J), if:

$$\mathsf{cost}_{(I,J)}(\Sigma) = \min_{\substack{\Sigma' \subset \mathcal{L} \\ \Sigma' \text{ finite}}} \mathsf{cost}_{(I,J)}\left(\Sigma'\right).$$

It is indeed possible that $\mathsf{cost}_{(I,J)}(\Sigma)$ is not defined. This is for instance the case for $\mathbf{T} = \{(R', 1)\}$, $\Sigma = \{\exists x\ R'(x)\}$ and $J = \varnothing$. However, this case is easily recognizable; in other cases we have a linear bound on the cost of a schema mapping:

Proposition 7. *There is a linear-time algorithm to check whether the cost of a schema mapping in $\mathcal{L}_{\text{tgd}}$ is defined with respect to a pair of instances. If it is defined, the cost is bounded by a linear function of the size of the data and the schema mapping itself.*

This linear bound is interesting, since we can imagine to use local search algorithms to find the tgd with minimum cost, *as soon as we are able to compute in an efficient way the cost of a tgd.* We shall see in Section 5, unfortunately, that even for a very restricted language, computing the cost of a tgd is **NP**-complete.

Example 8. Let us go back to the instances and schema mapping of Example 3, compute their respective cost, and see which one is optimal.

$$\mathsf{cost}_{(I,J)}(\Sigma_0) = 3 \cdot 2\,|J| = 30$$
$$\mathsf{cost}_{(I,J)}(\Sigma_1) = 3 + 2 + 3 \cdot 2 \cdot 2 = 17$$
$$\mathsf{cost}_{(I,J)}(\Sigma_2) = 3 + 4 \cdot 4 + 3 \cdot 2 = 25$$
$$\mathsf{cost}_{(I,J)}(\Sigma_3) = 4 + 4 + 3 \cdot 2 \cdot 4 = 32$$
$$\mathsf{cost}_{(I,J)}(\Sigma_4) = 2 + 4 + 3 \cdot 2 \cdot 4 = 30$$

It appears here that Σ_1 is the best of these schema mappings (and it can be shown that it is indeed optimal). As expected, Σ_2 is the second best.

The fact that Σ_4 has the same cost as Σ_0 is no coincidence, this is due to the choice we made for the cost of a ground fact.

At least on this simple example, our measure of cost seems reasonable. We will further justify it in Section 4.

The following decision problems arise naturally once given this notion of optimality. Each of them is defined for a given language $\mathcal{L}$, and we shall investigate their complexity in Section 5.

Validity. Given instances I, J, and a schema mapping $\Sigma \subset \mathcal{L}^*$, is Σ valid with respect to (I, J)?

Explanation. Given instances I, J, and a schema mapping $\Sigma \subset \mathcal{L}^*$, does Σ fully explain J with respect to I?

Zero-Repair. Given instances I, J, and a schema mapping $\Sigma \subset \mathcal{L}^*$, is $\mathsf{cost}_{(I,J)}(\Sigma)$ equal to $\mathsf{size}(\Sigma)$?

Cost. Given instances I, J, a schema mapping $\Sigma \subset \mathcal{L}$ and an integer $K \geqslant 0$, is $\mathsf{cost}_{(I,J)}(\Sigma)$ less than or equal to K?

Existence-Cost. Given instances I, J and an integer $K \geqslant 0$, does there exist a schema mapping $\Sigma \subset \mathcal{L}$ such that $\mathsf{cost}_{(I,J)}(\Sigma)$ is less than or equal to K?

Optimality. Given instances I, J, and a schema mapping $\Sigma \subset \mathcal{L}$, is it true that Σ is optimal with respect to (I, J)?

Note that VALIDITY, EXPLANATION and ZERO-REPAIR all consider formulas of the language of repairs $\mathcal{L}^*$. This is an important point that will be used in Proposition 10 to derive relationships between the six decision problems above.

4. JUSTIFICATION

In this section, we justify the definitions of the previous section by observing that, when instances I and J are derived from each other by elementary operators of the relational algebra, the optimal schema mapping, in $\mathcal{L}_{\text{tgd}}$, is the one that "naturally" describes this operator.

Let r, r' be instances of relations. We consider the following elementary operators of the relational algebra:

Projection. $\pi_i(r)$ denotes the projection of r along its ith attribute.

Selection. $\sigma_\varphi(r)$, where φ is a *conjunction* of equalities and negated equalities between an attribute of r and a constant, denotes the selection of r according to φ. Note that we allow neither identities between attributes of r (this is the role of the *join* operation), nor disjunctions (they may be expressed using a combination of selection and union).

Union. $r \cup r'$ is the union of r and r'.

Intersection. $r \cap r'$ is the intersection of r and r'.

Product. $r \times r'$ is the cross product of r and r'.

Join. $r \bowtie_\varphi r'$ is the join of r and r' according to φ, where φ is an equality between an attribute of r and an attribute of r'; φ is omitted when the context makes it clear.

The relationship between a database instance I and the instance J obtained from I using one of these operators can often be (partially) expressed in a natural way by a tgd or a set of tgds, where I is the source instance and J the target instance (and similarly when the source instance is expressed as the result of applying some operator to the target instance). For instance $\forall x\ R_1(x) \wedge R_2(x) \rightarrow R'(x)$ is naturally associated with the intersection operator. The only case when the relationship between I and J has no natural expression as a tgd is for the reciprocal of the union operator: If $I = R_1^J \cup R_2^J$, the natural formula for describing this relation is $\forall x\ R(x) \rightarrow R_1(x) \vee R_2(x)$, which is not a tgd since we do not allow disjunction. In some cases, as tgds are not powerful enough to express the relationship, or as some information is lost, the correspondence is only partial. For instance, $\forall x\ R(x) \rightarrow R'(x)$ is the most natural tgd for the operation $J = \sigma_\varphi(I)$, but the tgd does not fully describe the selection.

We now state that, using the notion of optimality of a schema mapping with respect to a pair of instances described in the previous section, and with some simple restrictions on the considered instances, the optimal schema mapping for a pair of instances obtained from each other with an operator of the relational algebra is precisely the schema mapping that is naturally associated with the operator. This justifies the choice of this notion of optimality, at least in these elementary contexts. We shall see in Section 6 other choices for the cost function, that might seem more natural at first, but that fail to satisfy the same property. For brevity's sake, we state this result in an informal way and illustrate it on the example of the join operator.

Theorem 9 (Informally stated)**.** *For any elementary operator γ of the relational algebra, the tgd naturally associated with this operator (when it exists) is optimal with respect to $(I, \gamma(I))$ (or $(\gamma(J), J)$, depending on the considered case), if some basic assumptions are fulfilled: the instances are not of trivial size and there is no simpler relation between attributes than the one described by the operator.*

Let us see what this means, and prove this result, for the join operator. Suppose $J = R_1^I \bowtie R_2^I$ (with R_1 and R_2 two binary relation symbols), and let us add the basic assumptions that $\pi_1(J) \neq \pi_2(J)$, $\pi_1(J) \neq \pi_3(J)$, $\pi_2(J) \neq \pi_3(J)$. We have:

$$\mathsf{cost}_{(I,J)}\left(\left\{\forall x \forall y \forall z\ R_1(x,y) \wedge R_2(y,z) \rightarrow R'(x,y,z)\right\}\right) = 7$$

since this tgd is valid and explains all facts of J. The cost of the empty schema mapping, $9\,|J|$, is greater since J is not empty. The only remaining relevant schema mappings with lesser size (of 5) have a single relation symbol R_1 or R_2 on the left-hand-side. But this means that they either predict two identical columns in J (this is incorrect, and has to be fixed in a repair of the schema mapping, whose additional cost is at least 2), or use an existential quantifier on the right-hand size, which also has to be repaired.

Now consider the case where $I = {R_1'}^J \bowtie {R_2'}^J$, and let us suppose all three attributes $\pi_i(I)$ disjoint.

$$\begin{aligned}\mathsf{cost}_{(I,J)}\left(\left\{\forall x \forall y \forall z\ R(x,y,z) \rightarrow R_1'(x,y) \wedge R_2'(y,z)\right\}\right) &= 7 + 6\left|\left\{\,(x,y) \in {R_1'}^J \mid \forall z\ (y,z) \notin {R_2'}^J\,\right\}\right| \\ &\quad + 6\left|\left\{\,(y,z) \in {R_2'}^J \mid \forall x\ (x,y) \notin {R_1'}^J\,\right\}\right|.\end{aligned}$$

$\mathsf{cost}_{(I,J)}(\varnothing) = 6\,|J|$ is greater than that as soon as I is not empty. As we assumed all three attributes of I disjoint, we can eliminate a number of schema mappings that do not produce any correct facts. The only remaining ones only have $R_1'(w_1, w_2)$ or $R_2'(w_2, w_3)$ terms on the right-hand size with those three variables either existentially quantified or appearing, respectively in the first, second or third position of a $R(w_1, w_2, w_3)$ atom on the left-hand side. None of these schema mappings can explain the facts that the schema mapping above does not explain, and existential quantifiers have to be accounted for in repairs.

These results could also be extended to the cases where we have relations of greater arity, but we would then require strong constraints, as the one we imposed for reciprocal join (that all attributes are disjoint), so as not to have any "hidden" relation between the different attributes. A weaker assumption that could be made is to use a notion of *Kolmogorov randomness* [19]: A database instance selected at random cannot have a description of length lower than its size, thanks to a simple counting argument. We can use such random instances to get a contradiction when we obtain a schema mapping that uses hidden relations between attributes of relations in the instance to have a lower cost than the natural schema mapping.

5. COMPLEXITY STUDY

We now proceed to a study of the computational complexity of the different problems identified in Section 3, for the different subsets of $\mathcal{L}_{\text{tgd}}$ that we presented in Section 2. We focus here on *combined complexity* (when K and Σ are part of the input to the problem in addition to I and J), since we are precisely

reasoning about the schema mappings themselves. We first describe general relationships between the different problems, before giving complexity results for $\mathcal{L}_{\text{tgd}}$, $\mathcal{L}_{\text{full}}$, $\mathcal{L}_{\text{facyc}}$ and $\mathcal{L}_{\text{acyc}}$, in that order (it might seem natural to analyze $\mathcal{L}_{\text{acyc}}$ before $\mathcal{L}_{\text{facyc}}$ but the proofs for the latter are slightly simpler, and will help to understand notions needed for the former). Cost and Existence-Cost will be discussed separately. We present at the end of the section *data complexity* results.

General Complexity Results. As the complexity of the different decision problems depends on the particular language considered, we add to the problem name a subscript identifying the considered language (say, Optimality$_{\text{tgd}}$ for the Optimality problem in $\mathcal{L}_{\text{tgd}}$).

We have the following elementary relationships between these problems, that can be used to derive complexity results for one problem from complexity results for another one.

Proposition 10. *For any language $\mathcal{L}$:*

1. Zero-Repair = Validity $\cap$ Explanation.
2. *There is a polynomial-time reduction of* Validity *to* Zero-Repair.
3. *There is a polynomial-time reduction of* Zero-Repair *to* Cost.
4. *Given an algorithm $\mathcal{A}$ for the* Zero-Repair *problem, and a polynomial-time algorithm for determining if a formula is in $\mathcal{L}$, there are non-deterministic algorithms for* Cost *and* Existence-Cost *that run by using once the algorithm $\mathcal{A}$, with an additional polynomial time cost.*
5. *Given an algorithm $\mathcal{A}$ for* Cost*, and a polynomial-time algorithm for determining if a formula is in $\mathcal{L}$, there is a non-deterministic algorithm for the complement of* Optimality *that runs by using a logarithmic number of times the algorithm $\mathcal{A}$, with an additional polynomial time cost.*
6. *If $\mathcal{L} \subseteq \mathcal{L}'$, for any problem among* Validity*,* Explanation*,* Zero-Repair *and* Cost*, there is a constant-time reduction from the problem in $\mathcal{L}$ to the problem in $\mathcal{L}'$.*

Note that for all languages considered here, there is a linear-time algorithm for determining if a formula is in this language; this is obvious for all except for $\mathcal{L}_{\text{acyc}}$ and $\mathcal{L}_{\text{facyc}}$, and an algorithm from [21] gives a linear-time algorithm for the acyclicity of hypergraphs.

In the next sections, we shall investigate in detail the complexity of the different problems in each of the identified subsets of $\mathcal{L}_{\text{tgd}}$, starting from $\mathcal{L}_{\text{tgd}}$ itself. A summary of all combined complexity results proved in the following, along with their direct consequences, is shown in Table 1.

Combined Complexity for TGDs. Let us first investigate the combined complexity of Validity and Explanation in $\mathcal{L}_{\text{tgd}}$.

Proposition 11.

1. Validity$_{\text{tgd}}$ *is* $\mathbf{\Pi_2^P}$*-complete.*
2. Explanation$_{\text{tgd}}$ *is in* ***NP****.*

Combined Complexity for Full TGDs. We now consider the language of full tgds, $\mathcal{L}_{\text{full}}$.

Proposition 12.

1. Validity$_{\text{full}}$ *is* ***coNP****-complete;*
2. Explanation$_{\text{full}}$ *and* Zero-Repair$_{\text{full}}$ *are* ***NP****-hard.*

Combined Complexity for Full Acyclic TGDs. We now look at the complexity of the same problems for $\mathcal{L}_{\text{facyc}}$. We shall need additional notions on *acyclic joins* from [2, 22]. Note first that an acyclic full tgd $\forall \mathbf{x}\ \varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$ that describes the relation between a pair of instances (I, J) can be seen, in the relational algebra, as a project-join expression over the source instance, $\pi_\psi(\bowtie_\varphi(I))$, φ expressing the join (which is, by hypothesis, acyclic) and ψ expressing the projection. Adding repaired formulas, of the form $\forall \mathbf{x}\ (\varphi(\mathbf{x}) \wedge \tau(\mathbf{x})) \rightarrow \psi(\mathbf{x})$, means adding an additional selection: $\pi_\psi(\sigma_\tau(\bowtie_\varphi(I)))$.

A *full reducer* of a join expression is a program which removes some tuples to the relations to be joined (by performing semi-joins) so that each relation can then be retrieved as a projection of the full join. Such a full reducer always exists in acyclic databases and can be obtained in polynomial time [4]. The full reducer itself runs in polynomial time. Finally, note that a join tree of an acyclic join can be obtained in linear time [21].

[22] proposes then Algorithm 1 for computing the result to a project-join expression on an acyclic database, that we reuse with slight modifications in our next proposition.

Algorithm 1 Result to a project-join expression on an acyclic database (after [22])

Input: An acyclic join expression φ, a project expression ψ, an instance I.
Output: $\pi_\psi(\bowtie_\varphi(I))$.

(a) Compute a full reducer of the relation instances, and apply it.
(b) Compute a join tree T of the acyclic expression. Each node of the tree initially contains the corresponding reduced relation instance.
(c) For each subtree of T with root r, compute recursively for each child r' of r the join of r with r', and project to the union of the variables appearing in ψ and the common variables of r and r'. Remove r' and replace node r with this result.

An important property of this algorithm is that, at all time, the size of the relation stored in node r of T is bounded by the original (reduced) size of r times the size of the final output. This means in particular that this algorithm computes in polynomial time the result to the project-join expression. Actually, the same algorithm can be applied when repaired formulas are considered, since the only selection performed is a conjunction of constraints (equality and negated equality) on a given variable: These selections can be pushed inside the join.

Proposition 13. Validity$_{\text{facyc}}$ *and* Explanation$_{\text{facyc}}$ *are in* ***PTIME****.*

Zero-Repair is then tractable in $\mathcal{L}_{\text{facyc}}$. One might hope that this tractability extends to Cost. Unfortunately, we now show the **NP**-hardness of Cost$_{\text{facyc}}$, even for a very simple schema mapping. For this purpose, we shall first need a quite general result on the minimal size of a vertex cover in a r-partite r-uniform hypergraph (for $r \geqslant 3$).

A hypergraph is r-partite if the set of vertices can be decomposed into an r-partition, such that no two vertices of the same partitioning subset are in a same hyperedge. It is r-uniform if all hyperedges have a cardinality of r. A vertex cover of a hypergraph is a subset X of the set of vertices, such that for every hyperedge e, at least one of the elements of

Table 1: Combined complexity results

	$\mathcal{L}_{\text{tgd}}$	$\mathcal{L}_{\text{full}}$	$\mathcal{L}_{\text{acyc}}$	$\mathcal{L}_{\text{facyc}}$
VALIDITY	$\mathbf{\Pi_2^P}$-complete	**coNP**-complete	**coNP**-complete	**PTIME**
EXPLANATION	**NP**-complete	**NP**-complete	**NP**-complete	**PTIME**
ZERO-REPAIR	$\mathbf{\Pi_2^P}$-complete	**DP**, (**co**)**NP**-hard	**DP**, (**co**)**NP**-hard	**PTIME**
COST	$\mathbf{\Sigma_3^P}$, $\mathbf{\Pi_2^P}$-hard	$\mathbf{\Sigma_2^P}$, (**co**)**NP**-hard	$\mathbf{\Sigma_2^P}$, (**co**)**NP**-hard	**NP**-complete
EXISTENCE-COST	$\mathbf{\Sigma_3^P}$, **NP**-hard	$\mathbf{\Sigma_2^P}$, **NP**-hard	$\mathbf{\Sigma_2^P}$, **NP**-hard	**NP**-complete
OPTIMALITY	$\mathbf{\Pi_4^P}$, (**co**)**NP**-hard	$\mathbf{\Pi_3^P}$, (**co**)**NP**-hard	$\mathbf{\Pi_3^P}$, (**co**)**NP**-hard	$\mathbf{\Pi_2^P}$, (**co**)**NP**-hard

e is in X. In regular graphs, VERTEX-COVER (determining whether there is a vertex cover of size $\leqslant K$) is one of the most known and useful **NP**-complete problems [11]. This obviously implies that VERTEX-COVER is **NP**-hard in general hypergraphs. Note that a 2-partite 2-uniform hypergraph is just a bipartite graph, and VERTEX-COVER in bipartite graphs is **PTIME**, thanks to Kőnig's theorem [6, 16] which states that the maximal number of matchings in a bipartite graph is the mimimum size of a vertex cover.

Lemma 14. *The problem of, given an r-partite r-uniform hypergraph $\mathcal{H}$ and a constant K, determining whether there exists a vertex cover in $\mathcal{H}$ of size less than or equal to K is **NP**-complete for $r \geqslant 3$.*

Proof. This problem is clearly in **NP**: Just guess a set of vertices of size less than or equal to K and check in polynomial time whether it is a vertex cover. For the hardness part, we prove the case $r = 3$; there is an obvious reduction from this case to the same problem for other values of r. We use a reduction from 3SAT.

Note that this result appears in [14], but the reduction presented there is not exhaustive (in particular, nothing is said about interdependencies between clauses, or the fact that the hypergraph is tripartite) and it is not clear whether the proof was indeed led to completion. We use here a proof inspired by the proof that 3-DIMENSIONAL-MATCHING is **NP**-hard in [11].

Let $\varphi = \bigwedge_{i=1}^{n} c_i$ be an instance of 3SAT, where the c_i are 3-clauses over some set $\mathbf{x}$ of variables. We build a tripartite 3-uniform hypergraph $\mathcal{H} = (V, E)$ (with vertex partition $V = V_1 \cup V_2 \cup V_3$) in the following way (see Figure 1 for an illustration when $\varphi = \neg z \vee x \vee y$). For each variable $x \in \mathbf{x}$, we add 12 nodes and 6 hyperedges to $\mathcal{H}$. 6 out of the 12 nodes are anonymous nodes which only appear in one hyperedge; they are denoted by $\bullet$. The other nodes are denoted x_1, x_2, x_3, $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$. Intuitively, all x_i's are in a minimum covering if and only if a valuation satisfying φ maps x_i to true (similarly with the $\bar{x}_i$'s and false). For each i, x_i and $\bar{x}_i$ belong to V_i. The so-called *local* hyperedges are shown in Table 2. Then, for each clause c_i, we add a single *global* hyperedge which contains the vertices corresponding to the variables appearing in c_i, while taking into account their position in the clause and whether they are negated. For instance, if $c_i = \neg z \vee x \vee y$, we add a hyperedge $(\bar{z}_1, x_2, y_3)$. This ensures that the hypergraph remains tripartite.

This reduction is polynomial. Let m be the cardinality of $\mathbf{x}$. We now show that φ is satisfiable if and only if there is a vertex cover in $\mathcal{H}$ of size less than or equal to $3m$ (or, equivalently, if there is a minimum vertex cover of size less than or equal to $3m$).

Table 2: Local edges used in the proof of Lemma 14

V_1	V_2	V_3
x_1	$\bullet$	$\bar{x}_3$
$\bullet$	x_2	$\bar{x}_3$
$\bar{x}_1$	x_2	$\bullet$
$\bar{x}_1$	$\bullet$	x_3
$\bullet$	$\bar{x}_2$	x_3
x_1	$\bar{x}_2$	$\bullet$

Suppose first that φ is satisfiable, and let ν be a valuation of $\mathbf{x}$ which satisfies φ. Let us consider the following set S of vertices of $\mathcal{H}$: For each $x \in \mathbf{x}$, we add to S, x_1, x_2 and x_3 if $\nu(x)$ is true, $\bar{x}_1$, $\bar{x}_2$ and $\bar{x}_3$ otherwise. S is of cardinality $3m$. Observe that S covers all local hyperedges and, since ν satisfies φ, all global hyperedges.

Suppose now that there is a minimum vertex cover S of size less than or equal to $3m$. Since anonymous vertices only appear in a single hyperedge, we can always assume that S does not contain any anonymous vertex (they can always be replaced by another vertex of the hyperedge). Let S_i be, for each $1 \leqslant i \leqslant m$, the subset of S_i containing only the vertices corresponding to the ith variable of $\mathbf{x}$. It is easy to see that $|S_i| \geqslant 3$ for all i, for all local hyperedges to be covered, which means that $|S_i| = 3$ since $|\bigcup S_i| \leqslant 3m$. S_i forms a vertex cover of the local sub-hypergraph corresponding to the ith variable of $\mathbf{x}$ (let us call it x) and must cover the hyperedges of this sub-hypergraph. But there are only two vertex covers of this sub-hypergraph of cardinality 3: Either S_i contains all x_k's, or it contains all $\bar{x}_k$'s. We consider the valuation ν of the variables in $\mathbf{x}$ which maps x to true in the first case, to false in the second. Then, since S is a vertex cover of $\mathcal{H}$, ν satisfies all the clauses of φ. □

We now use this lemma to prove the **NP**-hardness of $\text{COST}_{\text{facyc}}$.

Proposition 15. $\text{COST}_{\text{facyc}}$ *is **NP**-hard.*

Proof. We consider the case where we only allow negated equalities $x \neq c$, and no equalities $x = c$, on the left-hand side of repairs of tgds, with x a universally quantified variable, as the proof is clearer. Because of space constraints, discussion of the changes that have to be made in the general case is omitted.

We reduce the vertex cover problem in tripartite 3-uniform hypergraphs to $\text{COST}_{\text{facyc}}$. Let $\mathcal{H}$ be a tripartite 3-uniform hypergraph. We consider the following instance of $\text{COST}_{\text{facyc}}$:

- $\mathbf{S} = \{(R, 3)\}$ and R^I is the representation of $\mathcal{H}$ as a three-column table, where each row corresponds to an edge,

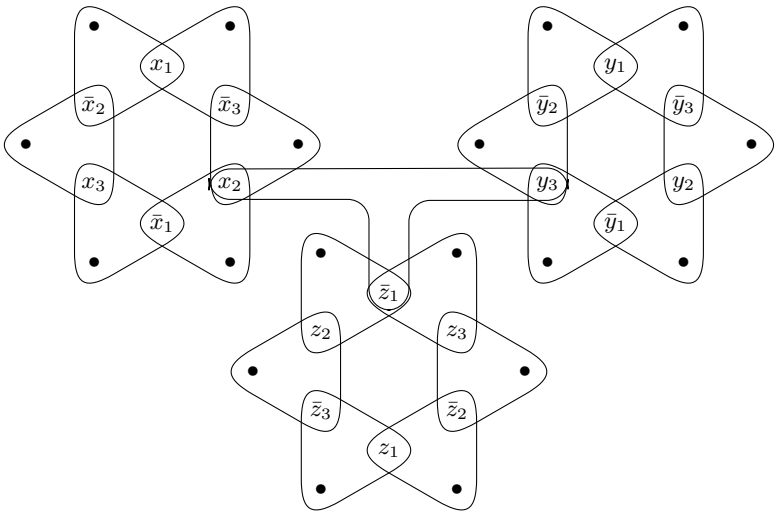

Figure 1: Example tripartite hypergraph corresponding to the 3SAT instance $\neg z \vee x \vee y$

and each column to one of the sets of the tripartition of $\mathcal{H}$;

- $\mathbf{T} = \{(R', 1)\}$ and $J = \varnothing$;
- $\Sigma = \{\forall x_1 \forall x_2 \forall x_3\ R(x_1, x_2, x_3) \rightarrow R'(x_1)\}$ (this is obviously an acyclic tgd).

As $J = \varnothing$, any schema mapping fully explains J. This also means that the only repairs of Σ to be considered are the ones that add a "$x_i \neq c_i$" term to the left-hand side of the single element of Σ. A repair of Σ has to "cancel" somehow with these additions each tuple of R^I. In other words, the cost of Σ is $\mathsf{size}(\Sigma) + 2r$, where r is the minimal number of conjuncts in a formula of the form $\bigwedge x_i \neq c_i$, such that this formula is false for all tuples of R^I. Such a formula expresses a vertex cover in $\mathcal{H}$, and $\mathcal{H}$ has a vertex cover of size less than or equal to K if and only if $\mathsf{cost}_{(I,J)}(\Sigma) \leqslant \mathsf{size}(\Sigma) + 2K$, which concludes the proof in this case. □

It is an open issue whether COST is in **PTIME** for the very restricted case when the schema mapping consists of a single full tgd with a single binary relation symbol appearing once in the left-hand side.

Combined Complexity for Acyclic TGDs. The last subset of $\mathcal{L}_{\text{tgd}}$ that we consider here is $\mathcal{L}_{\text{acyc}}$.

Proposition 16.

1. VALIDITY$_{\text{acyc}}$ *is* ***coNP****-complete.*
2. EXPLANATION$_{\text{acyc}}$ *and* ZERO-REPAIR$_{\text{acyc}}$ *are both* ***NP****-hard.*
3. EXPLANATION$_{\text{acyc}}$ *is in* ***PTIME*** if, for all existentially quantified variables y and for all constants c, there is at most one term $y = c$ appearing in each formula of the schema mapping. *This is in particular the case if the schema mapping is a subset of* $\mathcal{L}_{\text{acyc}}$ *instead of* $\mathcal{L}^*_{\text{acyc}}$.

Note that we use for both hardness results the repairs themselves to encode the instance of a **NP**-hard problem: Although the tgd itself is acyclic, its repairs are not. We could probably get polynomial algorithms for the same problems if we impose some acyclicity condition to repairs of a formula; this, however, would weaken our notion of optimality.

Table 3: Data complexity results

	$\mathcal{L}_{\text{tgd}}$, $\mathcal{L}_{\text{full}}$, $\mathcal{L}_{\text{acyc}}$, $\mathcal{L}_{\text{facyc}}$
VALIDITY	**PTIME**
EXPLANATION	**PTIME**
ZERO-REPAIR	**PTIME**
COST (K fixed)	**PTIME**
COST (Σ fixed)	**NP**, **NP**-hard for some Σ
EXISTENCE-COST	**PTIME**
OPTIMALITY	$\mathbf{\Pi_2^P}$, (**co**)**NP**-hard for some Σ

Combined Complexity of EXISTENCE-COST *and* OPTIMALITY. With the help of Lemma 14, we show the intractability of EXISTENCE-COST and OPTIMALITY, in all considered languages:

Proposition 17. EXISTENCE-COST *(respectively,* OPTIMALITY*) is* ***NP****-hard (respectively, both* ***NP****-hard and* ***coNP****-hard) in all the following languages:* $\mathcal{L}_{\text{tgd}}$, $\mathcal{L}_{\text{full}}$, $\mathcal{L}_{\text{acyc}}$, $\mathcal{L}_{\text{facyc}}$.

Data Complexity. As far as data complexity is concerned, the situation is simpler, since we do not have any difference in complexity for all four subsets of $\mathcal{L}_{\text{tgd}}$. The results are presented summarized in Table 3.

Proposition 18.

1. *If* Σ *is fixed,* VALIDITY$_{\text{rc}}$ *is in* ***PTIME****.*
2. *If* Σ *is fixed,* EXPLANATION$_{\text{tgd}}$ *is in* ***PTIME****.*
3. *If* K *is fixed,* COST$_{\text{tgd}}$ *and* EXISTENCE-COST$_{\text{tgd}}$ *are in* ***PTIME****.*
4. *For some fixed value of* Σ, COST$_{\text{facyc}}$ *is* ***NP****-hard.*
5. *For some fixed value of* Σ*, the problems* OPTIMALITY$_{\text{facyc}}$ *and* OPTIMALITY$_{\text{tgd}}$ *are* ***NP****-hard and* ***coNP****-hard.*

6. EXTENSION AND VARIANTS

We study in this section some extensions of our optimality notion to (i) formulas of the full relational calculus; (ii) other apparently simpler functions of cost.

Extension to the Relational Calculus. We can extend the definitions of Section 3 to the language $\mathcal{L}_{\text{rc}}$ of relational calculus, by the following adaptation of the notion of repair.

A repair of a schema mapping $\Sigma \subset \mathcal{L}_{\text{rc}}$ is a set of formulas obtained from Σ by a finite sequence of the following operations:

- Replacing in a formula of Σ a sub-formula $\forall \mathbf{x}\ \varphi(\mathbf{x}, \mathbf{y})$ (we also assume that φ does not start with a $\forall$ symbol, and that the sub-formula is not preceded by a $\forall$ symbol) by $\forall \mathbf{x}\ \tau(\mathbf{x}, \mathbf{z}) \rightarrow \varphi(\mathbf{x}, \mathbf{y})$ where $\mathbf{z}$ is the set of variables free in φ and τ is a boolean formula over terms $w = c$ of the following form:

$$\bigwedge_i \left(\left(\bigwedge_j z_{ij} = c'_{ij} \right) \rightarrow x_i \alpha_i c_i \right)$$

  with z_{ij} variables from $\mathbf{z}$, x_i variables from $\mathbf{x}$, α_i either $=$ or $\neq$, and c'_{ij} and c_i constants.

- Replacing in a formula of Σ a sub-formula $\exists \mathbf{x}\ \psi(\mathbf{x}, \mathbf{y})$ (we also assume that ψ does not start with a $\exists$ symbol, and that the sub-formula is not preceded by a $\exists$ symbol) by $\exists \mathbf{x}\ \psi(\mathbf{x}, \mathbf{y}) \wedge \tau'(\mathbf{x}, \mathbf{z})$ where $\mathbf{z}$ is the set of variables free in ψ and τ' is a boolean formula over terms $w = c$ of the following form:

$$\bigwedge_i \left(\left(\bigwedge_j z_{ij} = c'_{ij} \right) \rightarrow x_i = c_i \right)$$

  with z_{ij} variables from $\mathbf{z}$, x_i variables from $\mathbf{x}$, and c'_{ij} and c_i constants.

- Adding to Σ a ground fact $R(c_1 \dots c_n)$ with R a relation of the target schema of arity n, and $c_1 \dots c_n$ constants.

We can check that this definition amounts to the same as Definition 4 if we restrict ourselves to $\mathcal{L}_{\text{tgd}}$. We can then use the same definitions of the size and cost of a formula, and consider the same decision problems. We have the following complexity results:

Proposition 19.

1. VALIDITY$_{\text{rc}}$ *is* ***PSPACE****-complete;*
2. EXPLANATION$_{\text{rc}}$ *is co-recursively enumerable;*
3. EXPLANATION$_{\text{rc}}$ *and* ZERO-REPAIR$_{\text{rc}}$ *are not recursive.*

Interestingly, the computability of OPTIMALITY$_{\text{rc}}$ remains open. It seems to be a "harder" problem than ZERO-REPAIR$_{\text{rc}}$, but as there is no simple reduction between them, we cannot even be sure that OPTIMALITY$_{\text{rc}}$ is not recursive. We do not even know whether it is recursively enumerable or co-recursively enumerable (but COST$_{\text{rc}}$ and EXISTENCE-COST$_{\text{rc}}$ are both co-recursively enumerable because of the co-recursive enumerability of ZERO-REPAIR$_{\text{rc}}$).

Note that a related problem is studied in [10], where it is shown that determining whether there exists a schema mapping $\mathcal{L}_{\text{rc}}$ (without repairs) that is valid and explain all facts of the target instance is **coNP** and co-graph-isomorphism-hard. In the case of $\mathcal{L}^*_{\text{rc}}$, such a mapping obviously always exists since one can enumerate all ground facts of the target instance.

Variants of the Cost Function. The definition of repairs and cost that we presented in Section 3 may appear, at first, unnecessarily complicated. We argued in Section 4 for a justification of this notion by showing that it has nice properties with respect to instances that are derived from each other with elementary operations of the relational algebra. We consider in this section two alternative definitions of cost and optimality of a schema mapping with respect to database instances, and show that neither, although simpler and perhaps more intuitive, present the same properties and are thus adapted to our context.

We keep our notions of validity of a schema mapping, of full explanation of a database instance by a schema mapping, and of size of a schema mapping, and we want to consider alternative ways to characterize the *cost* of a given schema mapping. The first idea is to assign as the cost of a schema mapping the minimal number of tuples that have to be added or removed to the target instance J for the schema mapping to become valid and to fully explain J. (Each tuple may also be weighted by its arity, to get something closer to our original cost definition.) Thus, the cost of the empty schema mapping corresponds to the size of the target instance, as before, while the cost of a schema mapping that fully explains the target instance but also incorrectly explains some tuples is the (possibly weighted) number of such tuples. This sounds like a reasonable definition, but it presents an important problem: We lose the linear bound on the cost of a schema mapping in the size of the data and the schema mapping itself. Indeed, consider the following schema mapping, for a given n, where R is a source relation of arity 1 and R' a target relation of arity n:

$$\left\{ \forall x_1 \dots \forall x_n\ R(x_1) \wedge \dots \wedge R(x_n) \rightarrow R'(x_1, \dots, x_n) \right\}.$$

If J is empty, the cost of this schema mapping according to the definition given in this paragraph is $|I|^n$ (or $n\,|I|^n$ if we weight with the arity of the relations), which is exponential in the size of the schema mapping. This combinatorial explosion discourages all hopes of getting an optimal schema mapping by local search techniques. Besides, all the problems that we describe for the variant that we consider next also arise here.

An alternate definition of cost, close to the previous one but for which we still have a linear bound on the cost of a mapping is the following: The cost of a schema mapping Σ is the minimal number of tuples to add or remove from the source and target instances I and J so that Σ becomes valid and fully explains J. As before, we assume that we weight tuples by their arity; we could also choose to add an arbitrary constant weight to operations on J with respect to operations on I, or to deletion with respect to addition of tuples, without much difference. The linear bound is clear since we can just remove all tuples of I and of J for Σ to be valid and to fully explain J. However, there is still a fundamental problem with this definition, which can be seen by looking back at Section 4. We showed there that, for elementary operations of the relational algebra, the definition of optimality of Section 3 yielded the same as the intuitive tgds expressing these operations. This is not true any more here, however, in particular in the presence of selections and projections. For projections, this is due to the fact that a schema mapping that predicts existentially quantified tuples has a higher cost than the same schema mapping where these existentially quantified relation atoms are removed. We exhibit next a concrete example of database instances that illustrate the problem with selections.

Example 20. Let us consider instances I and J of the following schemata: $\mathbf{S} = \{(P, 2)\}$ and $\mathbf{T} = \{(P', 1)\}$, where: I contains a list of titles of publications (as first attribute) along with their kind: `article`, `book`, `report`, etc.; J contains a list of book titles. Let us assume that J and I contain the same book titles. In other words, $J = \pi_1(\sigma_{2=\texttt{book}}(I))$. It is quite natural to expect $\Sigma = \{\forall x \forall y\ P(x, y) \rightarrow P'(x)\}$ as the "optimal" schema mapping in the language of tgds for these database instances, and indeed, $\mathsf{cost}_{(I,J)}(\Sigma) = 5$ is minimal as soon as J is large enough and there is no hidden relation between the second attribute of I and J. Now, observe that with the variant proposed in the preceding paragraph, the cost will be: $3 + \min(2 \cdot (|I| - |J|), |J|)$, which is, in all cases when there are more publications of another kind than `book` (a common situation), greater than the cost of the empty schema mapping, which is then the optimal schema mapping for these instances.

Then, although our definition of optimality is a bit complex, it is much more adapted to the addressed problem than these simpler definitions, since it can capture such things as the worth of an existentially quantified relation atom, or the possibility of limiting the scope of a tgd with a simple selection.

7. CONCLUSION

We discussed a theoretical framework that addresses the problem of finding a schema mapping *optimal* with respect to a pair of database instances, based solely on the structure and occurrences of constants in the instances. We showed that this problem is both **NP**-hard and **coNP**-hard even for a very restricted language, namely full acyclic tuple-generating dependencies. This is not unexpected, since it is well known that such learning problems have high complexity even in very simple cases (see, for instance, [12] for ILP). Such a study is still useful since (i) it provides a formal framework for the discovery of schema mappings; (ii) complexity lower bounds are useful to detect the source of the complexity; (iii) complexity upper bounds often give practical algorithms.

There are a number of open theoretical issues, especially on the computability and precise complexity of Optimality, but the most obvious direction for future work would be to connect such a theoretical framework with practical heuristics and approximation algorithm; in particular, the relation to inductive logic programming has to be explored. We believe that this is an especially important problem, and that discovering and understanding hidden relations in data is one of the most fundamental tasks of artificial intelligence. Other problems of interest would be to improve our complexity upper bounds by generalizing the notion of acyclicity to that of bounded hypertree width [13], and to look at the same problems when some fixed set of preconditions on instances is given.

8. ACKNOWLEDGMENTS

We would like to thank Serge Abiteboul and Yann Ollivier for their input and feedback about this work.

9. REFERENCES

[1] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. PODS*, Philadelphia, United States, May 1999.

[2] C. Beeri, R. Fagin, D. Maier, A. Mendelzon, J. Ullman, and M. Yannakakis. Properties of acyclic database schemes. In *Proc. STOC*, Milwaukee, USA, May 1981.

[3] P. Bernstein. Applying model management to classical meta data. In *Proc. CIDR*, Asilomar, USA, Jan. 2003.

[4] P. A. Bernstein and D.-M. W. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM*, 28(1):25–40, 1981.

[5] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large Web sites. In *Proc. VLDB*, Roma, Italy, Sept. 2001.

[6] R. Diestel. *Graph Theory*. Springer, New York, USA, 2005.

[7] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. ICDT*, Siena, Italy, Jan. 2003.

[8] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Quasi-inverses of schema mapping. In *Proc. PODS*, Beijing, China, June 2007.

[9] R. Fagin, P. G. Kolaitis, W.-C. Tan, and L. Popa. Composing schema mappings: Second-order dependencies to the rescue. In *Proc. PODS*, Paris, France, June 2004.

[10] G. H. L. Fletcher. *On the data mapping problem*. PhD thesis, Indiana University, 2007.

[11] M. R. Garey and D. S. Johnson. *Computers And Intractability. A Guide to the Theory of NP-completeness*. W. H. Freeman, New York, USA, 1979.

[12] G. Gottlob, N. Leone, and F. Scarcello. On the complexity of some inductive logic programming problems. In *Proc. ILP*, Prag, Czech Republic, Sept. 1997.

[13] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. In *Proc. PODS*, Philadelphia, United States, May 1999.

[14] L. Ilie, R. Solis-Oba, and S. Yu. Reducing the size of NFAs by using equivalences and preorders. In *Combinatorial Pattern Matching*, Jeju Island, South Korea, June 2002.

[15] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proc. PODS*, Baltimore, Maryland, June 2005.

[16] D. Kőnig. *Theorie der endlichen und unendlichen Graphen*. Akademische Verlagsgesellschaft, Leipzig, Germany, 1936.

[17] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, USA, 1994.

[18] M. Lenzerini. Data integration: a theoretical perspective. In *Proc. PODS*, Madison, USA, June 2002.

[19] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, New York, USA, second edition, 1997.

[20] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[21] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[22] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB*, Cannes, France, Sept. 1981.