Designing and Refining Schema Mappings via Data Examples*

Bogdan Alexe
UCSC
abogdan@cs.ucsc.edu

Balder ten Cate
UCSC
balder.tencate@gmail.com

Phokion G. Kolaitis
UCSC & IBM Research - Almaden
kolaitis@cs.ucsc.edu

Wang-Chiew Tan
IBM Research - Almaden & UCSC
wangchiew@us.ibm.com

ABSTRACT

A schema mapping is a specification of the relationship between a source schema and a target schema. Schema mappings are fundamental building blocks in data integration and data exchange and, as such, obtaining the right schema mapping constitutes a major step towards the integration or exchange of data. Up to now, schema mappings have typically been specified manually or have been derived using mapping-design systems that automatically generate a schema mapping from a visual specification of the relationship between two schemas.

We present a novel paradigm and develop a system for the interactive design of schema mappings via data examples. Each data example represents a partial specification of the semantics of the desired schema mapping. At the core of our system lies a sound and complete algorithm that, given a finite set of data examples, decides whether or not there exists a GLAV schema mapping (i.e., a schema mapping specified by Global-and-Local-As-View constraints) that "fits" these data examples. If such a fitting GLAV schema mapping exists, then our system constructs the "most general" one.

We give a rigorous computational complexity analysis of the underlying decision problem concerning the existence of a fitting GLAV schema mapping, given a set of data examples. Specifically, we prove that this problem is complete for the second level of the polynomial hierarchy, hence, in a precise sense, harder than NP-complete. This worst-case complexity analysis notwithstanding, we conduct an experimental evaluation of our prototype implementation that demonstrates the feasibility of interactively designing schema mappings using data examples. In particular, our experiments show that our system achieves very good performance in real-life scenarios.

Categories and Subject Descriptors: H.2.5 [Database Management]: Heterogeneous Databases; H.2.4 [Database Management]: Systems - *relational databases*

General Terms: Algorithms, Design, Languages

Keywords: schema mappings, data examples, data exchange, data integration

*Research on this paper was supported by NSF grant IIS-0905276 and NSF Career Award IIS-0347065.

SIGMOD'11, June 12–16, 2011, Athens, Greece.

1. INTRODUCTION

A schema mapping is a high-level, declarative specification of the relationship between two database schemas, usually called the source schema and the target schema. By now, schema mappings are regarded as the fundamental building blocks in data exchange and data integration (see the surveys [5, 16, 17]). A crucial first step in these and other major data inter-operability tasks is the design and derivation of schema mappings between two database schemas.

Due to the complexity and scale of many real-world schemas, designing the "right" schema mapping can be a daunting task. This is further exacerbated by the fact that the formal specification of a schema mapping between such real-world schemas is typically long and difficult to grasp. For this reason, several different mapping-design systems have been developed to facilitate the process of designing schema mappings. These systems include Clio [15], HePToX [7], Microsoft's mapping composer [6], Altova Mapforce[1], and Stylus Studio[2]. The latter two systems directly produce transformations in the form of executable code, but can be viewed as implicitly constructing a schema mapping. Declarative specifications of schema mappings, as opposed to executable code, are more amenable to reasoning and analysis of data inter-operability tasks; furthermore, such specifications can be compiled into executable code for data exchange.

All mapping-design systems mentioned above adhere to the same general methodology for facilitating the process of designing schema mappings. Specifically, first a visual specification of the relationship between elements of the source and target schemas is solicited from the user (perhaps with the help of a schema matching module) and then a schema mapping is derived from the visual specification. However, several pairwise logically inequivalent schema mappings may be consistent with a visual specification, and, as noted in [4], differing schema mappings may be produced from the same visual specification, depending on which system is used. Thus, the user is still required to understand the formal specification in order to understand the semantics of the derived schema mapping.

In this paper, we present an alternative paradigm and develop a novel system for designing and refining schema mappings interactively via data examples. The intended users of our system are mapping designers who wish to design a schema mapping over a pair of source and target relational schemas. Furthermore, our system is tailored for schema mappings specified by *GLAV (Global-and-Local-As-View) constraints* (also known as *source-to-target tuple generating dependencies*, or *s-t tgds*). We call such schema mappings *GLAV schema mappings*. GLAV constraints have been extensively studied in the context of data exchange and data integration [16, 17]. The GLAV schema mappings contain, as important spe-

[1] http://www.altova.com/mapforce.html
[2] http://www.stylusstudio.com/xml_mapper.html

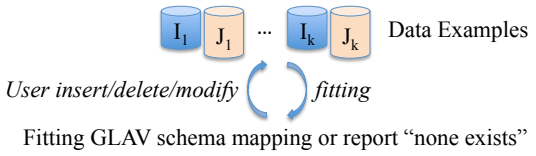

Figure 1: Workflow for interactive design and refinement of schema mappings via data examples.

cial cases, *Local-As-View (LAV)* schema mappings and *Global-As-View (GAV)* schema mappings. They are also used in such systems as Clio [15] and HePToX [7].

An important property of every GLAV schema mapping $\mathcal{M}$, first identified in [9], is that for every source instance I, there is a target instance J that is a *universal solution for I w.r.t. $\mathcal{M}$*. Intuitively, a universal solution of I w.r.t. $\mathcal{M}$ is a "most general" target instance of I that, together with I, satisfies the specifications of $\mathcal{M}$. Furthermore, a universal solution for I represents, in a precise technical sense, the entire space of solutions for I. As a result, universal solutions have become the preferred solutions and the standard semantics of data exchange. Our system makes systematic use of universal solutions, as we explain next.

The interaction between the user and our system begins with the user providing an initial finite set $\mathcal{E}$ of data examples, where a data example is a pair (I, J) consisting of a source instance and a target instance that conform to a source and target relational schema. Intuitively, each data example in $\mathcal{E}$ provides a partial specification of the semantics of the desired schema mapping. Furthermore, the user stipulates that, for each data example (I, J), the target instance J is a *universal solution for I* w.r.t. the desired schema mapping. The system responds by generating a GLAV schema mapping that *fits* the data examples in $\mathcal{E}$ or by reporting that no such GLAV schema mapping exists. Here, we say that a schema mapping $\mathcal{M}$ *fits a set $\mathcal{E}$* of data examples if for every data example $(I, J) \in \mathcal{E}$, the target instance J is a universal solution of the source instance I w.r.t. $\mathcal{M}$. The user is then allowed to modify the data examples in $\mathcal{E}$ and provide the system with another finite set $\mathcal{E}'$ of data examples. After this, the system tests whether or not there is a GLAV schema mapping that fits $\mathcal{E}'$ and, as before, returns such a GLAV schema mapping, if one exists, or reports that none exists, otherwise. The cycle of generating schema mappings and modifying data examples can be repeated until the user is satisfied with the schema mapping obtained in this way. We call this process the *interactive refinement* of a schema mapping via data examples. The workflow of our system is depicted in Figure 1.

The GLAV Fitting Algorithm At the core of our system lies a sound and complete algorithm for determining the existence of a fitting GLAV schema mapping, given a finite set of data examples. This GLAV fitting algorithm has several desirable properties. To begin with, if a fitting GLAV schema mapping exists, then the algorithm returns the *most general* fitting GLAV schema mapping. In other words, it returns a fitting GLAV schema mapping $\mathcal{M}$ such that for every alternative (non-equivalent) fitting GLAV schema mapping $\mathcal{M}'$, we have that $\mathcal{M}'$ logically implies $\mathcal{M}$. This is clearly the most natural choice among all the fitting GLAV schema mappings. The correctness of our algorithm is based on the following key technical result: the existence of a GLAV schema mapping that fits a given set $\mathcal{E}$ of data examples can be determined using a *homomorphism extension test*, which is an effective procedure that checks whether every homomorphism from the source instance of a data example $E_1 \in \mathcal{E}$ to that of another data example $E_2 \in \mathcal{E}$ can be extended to a homomorphism from the target instance of E_1 to the target instance of E_2. Actually, this homomorphism extension condition not only is a necessary and sufficient condition for the existence of a fitting GLAV schema mapping, but also implies the existence of a most general fitting GLAV schema mapping; furthermore, once the homomorphism extension test has been completed, the most general fitting GLAV schema mapping can be computed from the data examples $\mathcal{E}$ in linear time. The second and perhaps even more important feature of our algorithm is that our paradigm of designing schema mappings via data examples is complete, in the sense that every GLAV schema mapping can be produced by our algorithm. More precisely, for every GLAV schema mapping $\mathcal{M}$, there is a finite set of data examples $\mathcal{E}$ such that, when $\mathcal{E}$ is given as input, our GLAV fitting algorithm will produce a GLAV schema mapping that is logically equivalent to the desired one.

Complexity analysis The GLAV fitting algorithm that we described above runs in exponential time in the worst case. This is so because the homomorphism extension test entails examining, in the worst case, exponentially many homomorphisms. We show that, unless $\mathrm{P} = \mathrm{NP}$, this worst-case behavior is unavoidable. Indeed, we show that the underlying *GLAV fitting decision problem* (i.e., the problem of deciding whether, given a finite set of data examples, there is a fitting GLAV schema mapping) is Π_2^{P}-complete, where Π_2^{P} is the second level of the polynomial hierarchy. In particular, this means that, in all likelihood, the GLAV fitting decision problem is harder than NP-complete. Furthermore, we show that the *GLAV fitting verification problem* (i.e., the problem of deciding whether a given GLAV schema mapping fits a given set of data examples) is Π_2^{P}-complete as well.

Experimental Evaluation We built a prototype system and conducted an extensive experimental evaluation. In this evaluation, our system achieved very good performance on sets of data examples based on real-life scenarios, and thus confirmed the feasibility of interactively designing and refining schema mappings via data examples. The experimental results supporting this finding, as well as other observations regarding the behavior of our system on a variety of real-life and synthetic scenarios are discussed in Section 5.

1.1 An Illustration of Our System

Next, we will describe our system with a hypothetical execution of the workflow for Figure 1.

Suppose a user wishes to design a schema mapping between the source schema and target schema shown in the top-left corner of Figure 2. The source schema has two relations: Patient and Doctor, and the target schema has two relations: History and Physician. As a first step, the user specifies a single data example, shown in the first box, which essentially states that `Anna` is the doctor of patient `Joe`, whose health plan is `Plus`, and date-of-visit is `Jan`. In the target relation, there is a single fact that consolidates this information, omitting the name of the patient.

Based on this single data example, our system will infer that the desired schema mapping is the one shown on the right of the box. This schema mapping states that whenever a Patient tuple and Doctor tuple agree on the `pid` value (i.e., a natural join between Patient and Doctor), create a target tuple with the `pid`, `healthplan`, `date`, and `docid` values from Patient and Doctor. The user may choose to refine the data example further. (It may be because she realized that there was a typographical error in the specified data example, or she may have examined the schema mapping returned by the system and decide that further refinements are needed.)

After refinement, the data example shown in the second box is obtained. For this data example, the source instance remains unchanged, but the user has now modified the target instance to consist of two tuples: a History tuple and a Physician tuple which are "connected" through the value `N1`. Observe that the values `N1`

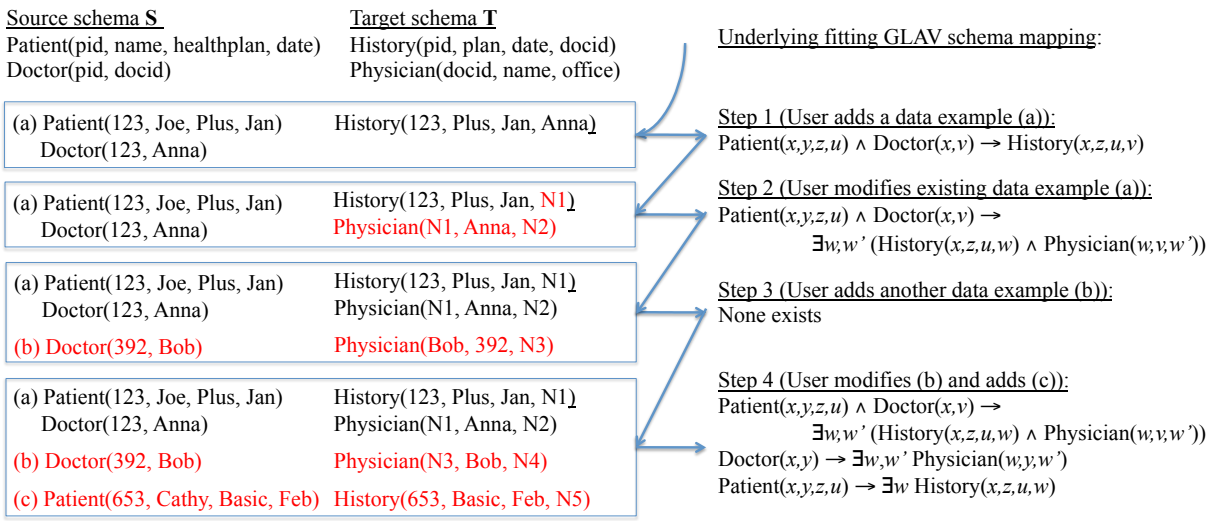

Figure 2: An example of Figure 1. Left: Each box is an input set of data examples. Right: Derived schema mapping, or none exists.

and N2 in the target instance do not occur among the values of the source instance and they, intuitively, represent unknown and possibly different values. Based on this single data example, our system infers the desired schema mapping shown on the right (i.e., under Step 2). Intuitively, this schema mapping states that information from the inner join of Patient and Doctor should be migrated to the target History and Physician relations, with appropriate nulls to represent unknown and possibly different values.

Further refinement steps can occur on the data examples. In the third box of Figure 2, the user adds a second data example (b) to the existing data example, and our system now reports that no schema mapping can "fit". This is because data example (b) describes a pattern of data migration that is inconsistent with data example (a): According to (b), every Doctor(pid,docid) fact in the source must have a corresponding Physician(docid,pid,office) fact in the target. Observe that the pid value is copied to the second column of the corresponding Physician fact. However, this is inconsistent with what (a) states, where a Doctor(pid, docid) has a corresponding Physician(_,docid,_) fact in the target, and docid gets copied to the second column of the corresponding Physician fact instead.

Finally, in the fourth box, the user modifies data example (b) and adds a third data example (c). Based on these data examples, our system reports the schema mapping shown under Step 4 on the right. Essentially, the schema mapping migrates information from the outer join of Doctor and Patient to the corresponding relations in the target.

1.2 Related Work

Data examples have already been used to illustrate different syntactic objects such as relational queries [18], dataflow programs [19], and schema mappings [2, 23]. These papers assume the existence of a syntactic object, and the goal is to produce informative examples that highlight particular features of the syntactic object under consideration. In addition, the systems of [2, 23] support the selection of a schema mapping among competing alternatives by offering the user a choice of examples, each of which illustrates one of the alternative schema mappings.

More recent work [3, 22] investigated whether or not a schema mapping can be uniquely characterized via a finite set of data examples of different types. Universal examples turned out to be the most robust and useful type of data example for capturing schema mappings, where a universal example is a data example (I, J) such that J is a universal solution for I w.r.t. a schema mapping. In particular, in [3] it was shown that for every LAV schema mapping, there is a finite set of universal examples that uniquely characterizes it among all LAV schema mappings. Furthermore, in [22], a necessary and sufficient condition was given for the unique characterizability of GAV schema mappings. The robustness and usefulness of universal examples are the reasons why we assume that for every input data example (I, J) to our system, the user intends that J is a universal solution for I w.r.t. the schema mapping that she has in mind.

Finally, there is a body of work on deriving a relational query or a schema mapping from one or more data examples [12, 13, 14, 21]. The work of [14] gave a theoretical framework and complexity results for the problem of deriving a schema mapping from a single data example. Even though a cost model is proposed to choose among schema mappings that can be used to describe a data example, they do not provide a direct algorithm for deriving a schema mapping from the data example. In [12], the goal is to heuristically derive a syntactic expression whose semantics coincide with a (possibly infinite) set of data examples. The papers [13, 21] are closer to our work, but both consider unirelational schemas only. The problem studied in [13] takes as input a finite set of pairs consisting of a single source relation and a single target relation, and asks whether or not there is a relational algebra expression that produces each of the target relations when evaluated on the corresponding source relation. In [21], a problem similar to the preceding one is studied, except that only a rather restricted class of relational algebra expressions (essentially unions of selections) is considered. In addition, the authors consider variations of the problem that take into account the size of the syntactic descriptions and notions of approximation.

2. PRELIMINARIES

Schemas and Instances A *relational schema* $\mathbf{R}$ is a finite sequence $(P_1, \ldots, P_k)$ of relation symbols, each of a fixed arity. An *instance* K over $\mathbf{R}$ is a sequence $(P_1^K, \ldots, P_k^K)$, where each P_i^K is a relation of the same arity as P_i. We shall often write P_i to denote both the relation symbol and the relation P_i^K that interprets it. Here, we assume that all values occurring in relations belong to some fixed infinite set dom of values. A *fact* of an instance K over a schema $\mathbf{R}$ is an expression $P(a_1, \ldots, a_m)$ such that P is a relation symbol of $\mathbf{R}$ and $(a_1, \ldots, a_m) \in P^K$. We denote by

$\mathrm{adom}(K)$ the *active domain* of an instance K, that is to say, the set of all values from dom occurring in facts of K.

Schema Mappings In what follows, we will typically consider two relational schemas, a *source schema* $\mathbf{S}$ and a *target schema* $\mathbf{T}$. We will often use I to denote an instance of $\mathbf{S}$ and J to denote an instance of $\mathbf{T}$, and we will refer to I as a *source instance* and to J as a *target instance*. A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ consisting of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, and a set Σ of constraints that are usually expressed as formulas in some logical formalism. In this paper, we focus on schema mappings specified by GLAV constraints. By definition, a *GLAV (Global-Local-As-View) constraint* is a first-order sentence φ of the form

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

where $\varphi(\mathbf{x})$ is a conjunction of atoms over $\mathbf{S}$, each variable in $\mathbf{x}$ occurs in at least one atom in $\varphi(\mathbf{x})$, and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over $\mathbf{T}$ with variables from $\mathbf{x}$ and $\mathbf{y}$. By an *atom* over a schema $\mathbf{R}$, we mean a formula $P(x_1, \ldots, x_m)$, where $P \in \mathbf{R}$ and $x_1, \ldots, x_m$ are variables, not necessarily distinct. For notational simplicity, we will often drop the universal quantifiers $\forall \mathbf{x}$ in the front of GLAV constraints.

GLAV constraints contain both GAV constraints and LAV constraints as important special cases. A *GAV (Global-As-View)* constraint is a GLAV constraint in which the right-hand side is a single atom, i.e., it is of the form

$$\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow P(\mathbf{x})) \ ,$$

while a *LAV (Local-As-View)* constraint is a GLAV constraint in which the left-hand side is a single atom, i.e., it is of the form

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

As an illustration of these notions, the constraint

$$\mathrm{Patient}(x, y, z, u) \wedge \mathrm{Doctor}(x, v) \rightarrow$$
$$\exists w, w'(\mathrm{History}(x, z, u, w) \wedge \mathrm{Physician}(w, v, w'))$$

is a GLAV constraint that is neither a GAV nor a LAV constraint. The constraint

$$\mathrm{Patient}(x, y, z, u) \wedge \mathrm{Doctor}(x, v) \rightarrow \mathrm{History}(x, z, u, v)$$

is a GAV constraint that is not a LAV constraint, while

$$\mathrm{Doctor}(x, y) \rightarrow \exists w, w'\mathrm{Physician}(w, y, w')$$

is a LAV constraint that is not a GAV constraint.

A *GLAV schema mapping* is a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ such that Σ is a finite set of GLAV constraints. Similarly, *GAV schema mappings* and *LAV schema mappings* are schema mappings in which Σ is a finite set of GAV constraints or, respectively, a finite set of LAV constraints.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping, I a source instance, and J a target instance. We say that J is a *solution of I w.r.t. $\mathcal{M}$* if $(I, J) \models \Sigma$, i.e., if (I, J) satisfies every constraint in Σ.

As a concrete example, suppose the source schema consists of the relation symbol $\mathrm{Patient}$, the target schema consists of the relation symbol $\mathrm{Physician}$, and the schema mapping $\mathcal{M}$ is specified by the LAV constraint $\mathrm{Doctor}(x, y) \rightarrow \exists w, w'\mathrm{Physician}(w, y, w')$. Consider the source instance

$$I = \{\mathrm{Doctor}(123, \mathrm{Anna}), \mathrm{Doctor}(392, \mathrm{Bob})\}$$

and the target instances

$$J_1 = \{\mathrm{Physician}(N1, \mathrm{Anna}, N2), \mathrm{Physician}(N3, \mathrm{Bob}, N4)\}$$
$$J_2 = \{\mathrm{Physician}(N1, \mathrm{Anna}, N2), \mathrm{Physician}(392, \mathrm{Bob}, N4)\}$$
$$J_3 = \{\mathrm{Physician}(N1, \mathrm{Anna}, N2)\}.$$

Both J_1 and J_2 are solutions for I w.r.t. $\mathcal{M}$, but J_3 is not. Observe that the solutions J_1 and J_2 contains values (namely, $N1$, $N2$, $N3$, $N4$) that do not occur in the active domain of the source instance I. Intuitively, these values can be thought of as labeled nulls.

Data Exchange, Homomorphisms, and Universal Solutions *Data exchange* is the following problem: given a schema mapping $\mathcal{M}$ and a source instance I, construct a solution J for I w.r.t. $\mathcal{M}$. As we just saw, a source instance may have more than one solution with respect to a given GLAV schema mapping. We will be interested expressly in *universal solutions*, which were identified in [9] as the preferred solutions for data exchange purposes. Universal solutions are defined in terms of *homomorphisms*, as follows.

Let I_1 and I_2 be two instances over the same relational schema $\mathbf{R}$. A *homomorphism* $h : I_1 \rightarrow I_2$ is a function from $\mathrm{adom}(I_1)$ to $\mathrm{adom}(I_2)$ such that for every fact $P(a_1, \ldots, a_m)$ of I_1, we have that $P(h(a_1), \ldots, h(a_m))$ is a fact of I_2. We write $I_1 \rightarrow I_2$ to denote the existence of a homomorphism $h : I_1 \rightarrow I_2$.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and let I be a source instance. A target instance J is a *universal solution* for I w.r.t. $\mathcal{M}$ if the following hold:

1. J is a solution for I w.r.t. $\mathcal{M}$.
2. For every solution J' of I w.r.t. $\mathcal{M}$, there is a homomorphism $h : J \rightarrow J'$ that is constant on $\mathrm{adom}(I) \cap \mathrm{adom}(J)$, that is to say, $h(a) = a$, for every value $a \in \mathrm{adom}(I) \cap \mathrm{adom}(J)$.

Intuitively, universal solutions are the "most general" solutions. Furthermore, in a precise sense, they represent the entire space of solutions (see [9]). For this reason, universal solutions have become the standard semantics for data exchange. Going back to our previous example, note that J_1 is a universal solution for I w.r.t the schema mapping $\mathcal{M}$ specified by the LAV constraint $\mathrm{Doctor}(x, y) \rightarrow \exists w, w'\mathrm{Physician}(w, y, w')$. In contrast, J_2 is not a universal solution for I w.r.t. $\mathcal{M}$, since there is no homomorphism from J_2 to J_1 that is constant on $\mathrm{adom}(I) \cap \mathrm{adom}(J_2)$.

For GLAV schema mappings $\mathcal{M}$ (and in fact for a much wider class of schema mappings), a variant of the *chase procedure* can be used to compute, given a source instance I, a *canonical* universal solution for I w.r.t. $\mathcal{M}$ in time bounded by a polynomial in the size of I (see [9]).

3. THE FITTING PROBLEM

In this section, we describe our algorithms for the fitting problem and its variants, prove the correctness of the algorithms, and obtain sharp complexity results for the fitting problem and its variants.

DEFINITION 3.1. Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ be a target schema. A *data example* is a pair (I, J) such that I is a source instance and J is a target instance.

We say that a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *fits a data example* (I, J) if J is a universal solution for I w.r.t. $\mathcal{M}$.

We say that $\mathcal{M}$ *fits a set $\mathcal{E}$ of data examples* if $\mathcal{M}$ fits every data example $(I, J) \in \mathcal{E}$.

Returning to the example of Section 2, the schema mapping specified by $\mathrm{Doctor}(x, y) \rightarrow \exists w, w'\mathrm{Physician}(w, y, w')$ fits the data example (I, J_1), but does not fit (I, J_2).

Data examples were considered in [3] as a means to illustrate and understand schema mappings. Several different notions of "fitting" were explored, including the just defined notion of fitting in terms of universal examples. This notion was shown in [3, 22] to be the most robust and useful one for capturing schema mappings via data examples.

One could also consider an alternative notion of "fitting" in which a schema mapping $\mathcal{M}$ fits a data example (I, J) if J is a solution

(not necessarily a universal solution) for I w.r.t. $\mathcal{M}$. The main reason we chose to use universal solutions in the notion of fitting is that universal solutions, being the most general solutions, are natural as data examples. Universal solutions contain just the information needed to represent the desired outcome of migrating data from source to target. In particular, they contain no extraneous or overspecified facts, unlike arbitrary solutions in general.

We now introduce two variants of the fitting problem, namely, the *fitting generation problem*, which is a function problem, and the *fitting decision problem*, which is the underlying decision problem. We begin with the latter.

DEFINITION 3.2. (GLAV Fitting Decision Problem)
Given a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, and a finite set $\mathcal{E}$ of data examples that conform to the schemas, the *GLAV Fitting Decision Problem* asks to tell whether or not there is a GLAV schema mapping $\mathcal{M}$ that fits $\mathcal{E}$.

DEFINITION 3.3. (GLAV Fitting Generation Problem)
Given a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, and a set $\mathcal{E}$ of data examples that conform to the schemas, the *GLAV Fitting Generation Problem* asks to construct a GLAV schema mapping $\mathcal{M}$ that fits $\mathcal{E}$, if such a schema mapping exists, or to report that "None exists", otherwise.

In a similar manner, we can define analogous fitting problems for GAV schema mappings and for LAV schema mappings. These problems will be briefly considered towards the end of this section. Our main focus, however, will be on the fitting problems for GLAV schema mappings.

Note that the alternative notion of "fitting" with solutions in place of universal solutions gives rise to trivial fitting decision and fitting generation problems since, in this case, the schema mapping with an empty set of constraints would "fit" every data example (I, J) (in fact, it would be the most general "fitting" schema mapping).

3.1 The GLAV Fitting Algorithm

In this section, we present our algorithm for the GLAV Fitting Generation Problem and establish its properties. The algorithm is given in Figure 3. We begin by discussing the main steps of the algorithm in detail.

As seen in Figure 3, our algorithm has two main steps. Given a finite set $\mathcal{E}$ of data examples, the first step of the algorithm tests for the existence of a GLAV schema mapping fitting $\mathcal{E}$. If no such fitting GLAV schema mapping exists, then the algorithm simply reports that none exists. Otherwise, the second step of the algorithm proceeds to construct a GLAV schema mapping that fits the set $\mathcal{E}$. Actually, the GLAV schema mapping constructed by our algorithm will turn out to have a number of additional desirable properties that we will document in what follows.

Homomorphism Extension Test Let (I, J) and (I', J') be two data examples. We say that a homomorphism $h : I \to I'$ *extends to a homomorphism* $\widehat{h} : J \to J'$ if for all $a \in adom(I) \cap adom(J)$, we have that $\widehat{h}(a) = h(a)$. The first step of the algorithm consists of the *homomorphism extension test*. Specifically, for every pair of data examples from the given set $\mathcal{E}$, the algorithm tests whether every homomorphism between the source instances of the two examples extends to a homomorphism between the corresponding target instances. If this homomorphism extension test fails, the algorithm reports that no GLAV schema mapping fitting the set $\mathcal{E}$ exists. Otherwise, it proceeds to the next step.

Constructing a Fitting Canonical GLAV Schema Mapping In the second step of the algorithm, we make crucial use of the notion of a *canonical GLAV schema mapping* associated with a set of data examples. Intuitively, this is analogous to the familiar notion of a *canonical conjunctive query* associated with a database instance. If (I, J) is a data example, then the *canonical GLAV constraint of* (I, J) is the GLAV constraint $\forall \mathbf{x}(q_I(\mathbf{x}) \to \exists \mathbf{y} q_J(\mathbf{x}, \mathbf{y}))$, where $q_I(\mathbf{x})$ is the conjunction of all facts of I (with each value from the active domain of I replaced by a universally quantified variable from $\mathbf{x}$) and $q_J(\mathbf{x}, \mathbf{y})$ is the conjunction of all facts of J (with each value from $\mathrm{adom}(J) \setminus \mathrm{adom}(I)$ replaced by an existentially quantified variable from $\mathbf{y}$). If $\mathcal{E}$ is a finite set of data examples over a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$, then the *canonical GLAV schema mapping of* $\mathcal{E}$ is the schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ consists of the canonical GLAV constraints of the data examples in $\mathcal{E}$. The second step of the algorithm amounts simply to computing the canonical GLAV schema mapping of the given set of data examples. Notice that this step takes time linear in the size of the given set $\mathcal{E}$ of data examples.

Algorithm: GLAV Fitting
Input: A source schema $\mathbf{S}$ and a target schema $\mathbf{T}$
A finite set $\mathcal{E}$ of data examples $(I_1, J_1) \ldots (I_n, J_n)$ over $\mathbf{S}, \mathbf{T}$
Output: Either a fitting GLAV schema mapping or 'None exists'

// Homomorphism Extension Test:
// Test for existence of a fitting GLAV schema mapping
for all $i, j \leq n$ **do**
  for all homomorphisms $h : I_i \to I_j$ **do**
    if not(h extends to a homomorphism $\widehat{h} : J_i \to J_j$) **then**
      fail('None exists')
    end if
  end for
end for;
// Construct a fitting canonical GLAV schema mapping
$\Sigma := \emptyset$;
for all $i \leq n$ **do**
  add to Σ the canonical GLAV constraint of (I_i, J_i)
end for;
return ($\mathbf{S}$, $\mathbf{T}$, Σ)

Figure 3: The GLAV Fitting Generation Algorithm

It is important to point out that the canonical GLAV schema mapping of a given set of data examples need *not* fit this set of examples; as a matter of fact, this is what makes the GLAV fitting generation problem interesting and nontrivial. To illustrate this, consider the set $\mathcal{E}$ consisting of the data examples $(\{S(a, b)\}, \{T(a)\})$ and $(\{S(c, c)\}, \{U(c, d)\})$. The canonical GLAV schema mapping of $\mathcal{E}$ is specified by the GLAV constraints $\forall xy(S(x, y) \to T(x))$ and $\forall x(S(x, x) \to \exists z U(x, z))$. This schema mapping does not fit $\mathcal{E}$, as the second data example violates the first constraint. Note also that our homomorphism extension test in the first step of the algorithm would detect this: the homomorphism h that maps $S(a, b)$ to $S(c, c)$ does not extend to any target homomorphism from $T(a)$ to $U(c, d)$. Hence, in this case, our algorithm will terminate after the first step and report that "None exists".

Correctness The following result establishes the correctness of the GLAV fitting generation algorithm in Figure 3.

THEOREM 3.4. *Let $\mathcal{E}$ be a finite set of data examples. The following statements are equivalent:*

1. *The canonical GLAV schema mapping of $\mathcal{E}$ fits $\mathcal{E}$.*
2. *There is a GLAV schema mapping that fits $\mathcal{E}$.*
3. *For all $(I, J), (I', J') \in \mathcal{E}$, every homomorphism $h : I \to I'$ extends to a homomorphism $\widehat{h} : J \to J'$.*

PROOF HINT. The proof proceeds in a round-robin fashion. The implication $1 \Rightarrow 2$ is immediate. The implication $2 \Rightarrow 3$ is proved

using a monotonicity property of the chase. Finally, the implication $3 \Rightarrow 1$ follows from the construction of the canonical GLAV schema mapping. The basic idea is that, if (I, J) and (I', J') are data examples and $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y}))$ is the canonical GLAV constraint of (I, J), then each homomorphisms $h : I \rightarrow I'$ corresponds to an assignment for $\mathbf{x}$ under which $\phi(\mathbf{x})$ is true in I', and each extension $\widehat{h} : J \rightarrow J'$ of h corresponds to an assignment for $\mathbf{x}, \mathbf{y}$ under which $\psi(\mathbf{x}, \mathbf{y})$ is true in J'. $\square$

The last condition in Theorem 3.4 is the homomorphism extension test used in the first step of the algorithm. Theorem 3.4 shows that this is a necessary and sufficient condition for determining whether GLAV schema mapping fitting $\mathcal{E}$ exists. Furthermore, this condition is also a necessary and sufficient condition for determining whether the canonical GLAV schema mapping of $\mathcal{E}$ fits $\mathcal{E}$. Hence, the algorithm is correct in that it reports that no fitting GLAV schema mapping exists, if there is none, and it returns a fitting GLAV schema mapping, whenever there is one.

Most General Fitting GLAV Schema Mapping Given a finite set $\mathcal{E}$ of data examples, there may be a multitude of GLAV schema mappings that fit it. For example, both $R(x, y) \rightarrow P(x, y)$ and $R(x, x) \rightarrow P(x, x)$ fit the data example $(\{R(a, a)\}, \{P(a, a)\})$. In this case, the GLAV fitting algorithm will return the latter mapping $R(x, x) \rightarrow P(x, x)$. We will see that our GLAV fitting algorithm does this for a very good reason. Specifically, if a fitting GLAV schema mapping exists, then our algorithm returns *the most general* fitting GLAV schema mapping.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma)$ be two schema mappings over the same source and target schemas. We say that $\mathcal{M}$ is *more general than* $\mathcal{M}'$ if Σ' logically implies Σ, i.e., if for every data example (I, J) such that (I, J) satisfies Σ', we have that (I, J) also satisfies Σ.

THEOREM 3.5. *Let $\mathcal{E}$ be a finite set of data examples. If there is a GLAV schema mapping that fits $\mathcal{E}$, then the canonical GLAV schema mapping of $\mathcal{E}$ is the most general GLAV schema mapping that fits $\mathcal{E}$. In other words, the canonical GLAV schema mapping is more general than any GLAV schema mapping that fits $\mathcal{E}$.*

Theorems 3.4 and 3.5 imply that if a fitting GLAV schema mapping exists for a given set $\mathcal{E}$ of data examples, then our GLAV fitting algorithm returns the most general GLAV schema mapping that fits $\mathcal{E}$. Note that this most general schema mapping is unique up to logical equivalence.

The most general schema mapping produced by our GLAV fitting generation algorithm has size linear in the size of the input set of data examples. This linear bound on the size of the most general schema mapping cannot be improved in general. To see this, for every integer $n > 2$, consider the data example (I_n, J_n), where $I_n = \{E(1, 2), E(2, 3), \ldots, E(n-1, n)\}$ and $J_n = \{F(1, n)\}$. Then, for every n, the most general schema mapping fitting (I_n, J_n) is the schema mapping $\mathcal{M}_n$ specified by the single s-t tgd $E(x_1, x_2) \wedge E(x_2, x_3) \wedge \ldots \wedge E(x_{n-1}, x_n) \rightarrow F(x_1, x_n)$. It is not hard to see that no schema mapping logically equivalent to $\mathcal{M}_n$ has size less than that of $\mathcal{M}_n$. This is so because, intuitively, a schema mapping $\mathcal{M}$ that is logically equivalent to $\mathcal{M}_n$ must contain an s-t tgd where the left-hand-side has at least $n - 1$ atoms.

The most general GLAV schema mapping $\mathcal{M}$ enjoys a robustness property in that $\mathcal{M}$ can be used to derive the "certain certain answers", as we describe next. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and let q be a query over the target schema $\mathbf{T}$. If I is a source instance, then the *certain answers of q on I w.r.t. $\mathcal{M}$*, denoted by $\mathrm{certain}_{\mathcal{M}}(q, I)$, are defined by

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

The certain answers are the standard semantics of target queries in data exchange [9]. Now, it is not hard to show that if $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ are two GLAV schema mappings such that $\mathcal{M}$ is more general than $\mathcal{M}'$, then for every source instance I and for every target conjunctive query q, we have that $\mathrm{certain}_{\mathcal{M}}(q, I) \subseteq \mathrm{certain}_{\mathcal{M}'}(q, I)$. In turn, this observation implies the following result.

COROLLARY 3.6. *Let $\mathcal{E}$ be a finite set of data examples for which there exists a GLAV schema mapping that fits it. Let $\mathcal{G}$ be the set of all GLAV schema mappings that fit $\mathcal{E}$ and let $\mathcal{M} \in \mathcal{G}$ be the canonical GLAV schema mapping of $\mathcal{E}$. Then, for every source instance I and every target conjunctive query q, we have that* $\mathrm{certain}_{\mathcal{M}}(q, I) = \bigcap_{\mathcal{M}' \in \mathcal{G}} \mathrm{certain}_{\mathcal{M}'}(q, I)$.

Informally, Corollary 3.6 asserts that if $\mathcal{E}$ is a finite set of data examples, then the certain answers of a target conjunctive query w.r.t. to the canonical GLAV schema mapping of $\mathcal{E}$ are the "certain certain answers" over all GLAV schema mappings that fit $\mathcal{E}$.

Completeness for GLAV-Schema-Mapping Design A more important beneficial feature of the GLAV fitting algorithm that we wish to highlight here is that it is *complete for GLAV-schema-mapping design*, in the sense expressed by the next result.

THEOREM 3.7. *For every GLAV schema mapping $\mathcal{M}$, there is a finite set of data examples $\mathcal{E}_{\mathcal{M}}$, such that, when given $\mathcal{E}_{\mathcal{M}}$ as input, the GLAV fitting algorithm returns a schema mapping that is logically equivalent to $\mathcal{M}$.*

In other words, every GLAV schema mapping can be produced (up to logical equivalence) by our GLAV fitting algorithm. The proof of Theorem 3.7 is based on the notion of a *canonical set of examples for a GLAV schema mapping* (cf. also Section 5). Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a GLAV schema mapping. For every GLAV constraint $\sigma \in \Sigma$ of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$, let I_σ be the canonical instance of $\phi(\mathbf{x})$ and let J_σ be the universal solution of I_σ w.r.t. $\mathcal{M}$ obtained by chasing I_σ with Σ. We define the *canonical set of examples of* $\mathcal{M}$ to be the set of all examples (I_σ, J_σ) such that $\sigma \in \Sigma$. Now, it can be verified that, for every GLAV schema mapping $\mathcal{M}$, the canonical schema mapping of the canonical set of examples of $\mathcal{M}$ is logically equivalent to $\mathcal{M}$; consequently, Theorem 3.7 holds.

Complexity Analysis As mentioned earlier, the second step of the GLAV fitting algorithm is linear in the size of the input set of data examples. In contrast, the first step of the GLAV fitting algorithm can be exponential, since the number of homomorphisms between two database instances can be exponential; thus, the GLAV fitting algorithm runs in exponential time in the worst case. Furthermore, this worst-case behavior is realized on every input for which a fitting GLAV schema mapping exists, since, in such cases, it is not possible to terminate early the task of verifying that every homomorphism between the source instances of every pair of data examples indeed extends to a homomorphism on the respective target instances. However, the GLAV fitting algorithm needs a polynomial amount of memory only; this is so because the size of each particular homomorphism is polynomial, and it is possible to enumerate all homomorphisms between two instances using only a polynomial amount of memory.

Assuming that P $\neq$ NP, this exponential worst-case running time is unavoidable. Indeed, every algorithm that solves the GLAV

Fitting Generation Problem also solves the GLAV Fitting Decision Problem, which is NP-hard. In fact, our next result shows that the GLAV Fitting Decision Problem is complete for the second level Π_2^{p} of the polynomial hierarchy PH, hence, in all likelihood, it is harder than NP-complete. Recall that Π_2^{p} (also known as $\mathrm{coNP}^{\mathrm{NP}}$) is the class of all decision problems that can be solved by a coNP-algorithm using an NP-complete problem as an oracle. Furthermore, $\mathrm{NP} \cup \mathrm{coNP} \subseteq \Pi_2^{\mathrm{p}} \subseteq \mathrm{PSPACE}$ (see [20]).

THEOREM 3.8. *The GLAV Fitting Decision Problem is Π_2^{p}-complete.*

The GLAV fitting algorithm in Figure 3 actually shows that the GLAV fitting decision problem is in the class Π_2^{p}. Indeed, the homomorphism extension test in the first step involves a universal quantification over homomorphisms followed by an existential quantification over homomorphisms; this is a Π_2^{p}-computation. The Π_2^{p}-hardness of the GLAV fitting decision problem is proved via a reduction from the evaluation problem for quantified Boolean formulas of the form $\forall \mathbf{x} \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$, where $\psi(\mathbf{x}, \mathbf{y})$ is a 3CNF Boolean formula. Perhaps surprisingly, the lower bound holds even for inputs consisting of a single data example over fixed schemas; roughly speaking, for a given quantified Boolean formula of the form $\forall \mathbf{x} \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$, an example (I, J) is constructed in such a way that each nontrivial endomorphism of I corresponds to a valuation for $\mathbf{x}$, and J has facts describing the clauses of $\psi(\mathbf{x}, \mathbf{y})$.

We conclude the complexity analysis by mentioning in passing a consequence that Theorem 3.8 has for another variant of the fitting problem.

DEFINITION 3.9. (GLAV Fitting Verification Problem) Given a GLAV schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and a finite set $\mathcal{E}$ of data examples, the *GLAV Fitting Verification Problem* asks to tell whether or not $\mathcal{M}$ fits $\mathcal{E}$.

COROLLARY 3.10. *The GLAV Fitting Verification Problem is Π_2^{p}-complete.*

In other words, testing for the existence of a fitting GLAV schema mapping is no harder than testing whether a given GLAV schema mapping fits a given set of data examples. The lower bound follows directly from Theorems 3.4 and 3.8. The proof of the upper bound uses (among other things) the fact that GLAV constraints belong to the universal-existential fragment of first-order logic.

3.2 GAV and LAV Schema Mappings

GAV schema mappings and LAV schema mappings are syntactically simpler types of GLAV schema mappings that enjoy several additional good properties and are supported by many data-exchange and data-integration systems. In this section, we consider the GAV fitting generation problem and the LAV fitting generation problem, i.e., the problem that asks, given a finite set $\mathcal{E}$ of data examples, to generate a GAV (or a LAV) schema mapping that fits $\mathcal{E}$. Note that even if, for a given set of data examples, there exists a fitting GAV or fitting LAV schema mapping, there is no guarantee that a GLAV fitting generation algorithm (and, in particular, our GLAV fitting algorithm) will produce such a GAV or a LAV schema mapping. In what follows, we discuss how our GLAV fitting generation algorithm can be adapted in order to solve the GAV fitting generation problem and the LAV fitting generation problem.

GAV schema mappings It is known from [22] that the GAV fitting decision problem is DP-complete, and that it is NP-complete for inputs consisting of ground data examples. We say a data example (I, J) is *ground* if $adom(J) \subseteq adom(I)$. In other words, the instance J consists entirely of values from I.

THEOREM 3.11 ([22]). *The GAV fitting decision problem is* DP*-complete. It is* coNP*-complete if the input consists of ground data examples.*

As in the GLAV case, the lower bounds hold already for inputs consisting of a single example, over a fixed source schema and target schema. Recall that DP is the class of problems that can be defined as the conjunction of an NP-problem and a coNP-problem. It is known that $\mathrm{NP} \cup \mathrm{coNP} \subseteq \mathrm{DP} \subseteq \Pi_2^{\mathrm{p}}$, and it is widely believed that both containments are proper. In particular, just as in the GLAV case, we cannot hope for a polynomial-time algorithm that solves the GAV fitting generation problem.

We present here a concrete algorithm for solving the GAV fitting generation problem. The algorithm is given in Figure 4. It is a variation of the algorithm for the GLAV fitting generation problem given in Figure 3. Comparing the two algorithms, one can see that there are three essential differences.

The first difference is that the algorithm checks that each data example is essentially ground. Recall that we say a data example is *ground* if $adom(J) \subseteq adom(I)$. A data example is *essentially ground* if there is a homomorphism $h : J \to J$ such that $h(a) = a$ for all $a \in adom(I)$, and such that $rng(h) \subseteq adom(I)$. This is equivalent to saying that the core of J (as defined in [11], where the values from $adom(I)$ are viewed as constants, and the values from $adom(J) \setminus adom(I)$ are viewed as labeled nulls) contains only values from $adom(I)$.

The second difference is that the homomorphism extension test is now simplified. The test simply checks whether every homomorphism between the source instances of two examples is also a partial homomorphism (i.e., a partial function that preserves facts) between the corresponding target instances. Another way to say the same thing is that for every pair of examples $(I, J), (I', J')$, every homomorphism $h : I \to I'$ is a homomorphism from $J_{\downarrow}$ to $J'_{\downarrow}$, where $J_{\downarrow}$ is the subinstance of J containing only values from $\mathrm{adom}(I)$, and $J'_{\downarrow}$ is the subinstance of J' containing only values from $\mathrm{adom}(I')$.

Third, instead of constructing the *canonical GLAV schema mapping*, the algorithm constructs the *canonical GAV schema mapping*. For any set $\mathcal{E}$ of examples, the *canonical GAV schema mapping of* $\mathcal{E}$ is defined in the same way as the canonical GLAV schema mapping of $\mathcal{E}$, but ignoring all target facts containing nulls. In other words, the canonical GAV schema mapping of $\mathcal{E}$ is the canonical GLAV schema mapping of $\{(I, J_{\downarrow}) \mid (I, J) \in \mathcal{E}\}$, where $J_{\downarrow}$ consists of all facts of J that contain only values from $\mathrm{adom}(I)$. Note that this is indeed a GAV schema mapping.

The correctness of our algorithm is given by the following result.

THEOREM 3.12. *Let $\mathcal{E}$ be a finite set of data examples. The following are equivalent:*

- *The canonical GAV schema mapping of $\mathcal{E}$ fits $\mathcal{E}$.*
- *There is a GAV schema mapping that $\mathcal{E}$.*
- *Each data example $(I, J) \in \mathcal{E}$ is essentially ground, and for all $(I, J), (I', J') \in \mathcal{E}$, every homomorphism $h : I \to I'$ is a partial homomorphism $h : J \to J'$.*

The proof of Theorem 3.12 is along the same lines as the one of Theorem 3.4, but using also the fact that, for GAV schema mappings, every source instance has a ground universal solution.

As in the GLAV case, the GAV schema mapping produced by our algorithm is the most general fitting GAV schema mapping.

THEOREM 3.13. *Let $\mathcal{E}$ be a finite set of data examples for which there exists a fitting GAV schema mapping. Then the canonical GAV schema mapping of $\mathcal{E}$ is the most general fitting GAV schema mapping of $\mathcal{E}$.*

Algorithm: GAV fitting
Input: A source schema $\mathbf{S}$ *and a target schema* $\mathbf{T}$
A finite set of data examples $(I_1, J_1) \ldots (I_n, J_n)$ *over* $\mathbf{S}, \mathbf{T}$
Output: either a fitting GAV schema mapping or 'None exists'

// Test that each data example is essentially ground
for all $i \leq n$ **do**
  if not(there is a homomorphism $h : J_i \rightarrow J_i$ such that $h(a) = a$ for all $a \in adom(I)$ and $rng(h) \subseteq adom(I)$) **then** fail('None exists');
  end if
end for;
// Simplified Homomorphism Extension Test:
// test for the existence of a fitting GAV schema mapping
for all $i, j \leq n$ **do**
  for all homomorphisms $h : I_i \rightarrow I_j$ **do**
    if not(h is a partial homomorphism $h : J_i \rightarrow J_j$) **then** fail('None exists')
    end if
  end for
end for;
// Construct fitting canonical GAV schema mapping
$\Sigma := \emptyset$;
for all $i \leq n$ **do**
  add to Σ the canonical GAV constraint of (I_i, J_i)
end for;
return ($\mathbf{S}$, $\mathbf{T}$, Σ)

Figure 4: The GAV Fitting Generation Algorithm

Theorem 3.13 is implicit in [22], even though the notion of a canonical GAV schema mapping was not introduced there.

Furthermore, just as the GLAV fitting generation algorithm is complete for GLAV-schema-mapping design (cf. Theorem 3.7), in the same way the GAV fitting generation algorithm can be shown to be complete for GAV-schema-mapping design.

THEOREM 3.14. *For every GAV schema mapping* $\mathcal{M}$*, there is a finite set of data examples* $\mathcal{E}_\mathcal{M}$*, such that, when given* $\mathcal{E}_\mathcal{M}$ *as input, the GAV fitting algorithm returns a schema mapping that is logically equivalent to* $\mathcal{M}$*.*

LAV schema mappings For LAV schema mappings, the situation is a bit different. Unlike in the case of GLAV schema mappings or GAV schema mappings, there is no natural notion of a *canonical LAV* schema mapping, for a given set of examples. Furthermore, a set of examples for which there are fitting LAV schema mappings may not have a most general fitting LAV schema mapping. Indeed, consider the single example (I, J) where $I = \{P(a), Q(a)\}$ and $J = \{R(a)\}$. It is not hard to see that there are two incomparable maximally general fitting LAV schema mappings, specified by the LAV constraints $\forall x(P(x) \rightarrow R(x))$ and $\forall x(Q(x) \rightarrow R(x))$ respectively. Nevertheless, the LAV fitting generation problem can be solved in exponential time. For lack of space, we omit a detailed description of the algorithm, and state only the following result:

THEOREM 3.15. *The LAV fitting decision problem is NP-complete.*

The upper bound is proved by giving a non-deterministic polynomial-time algorithm that has an accepting run if and only if there is a LAV fitting decision problem, and that, moreover, computes a fitting LAV schema mapping if it exists. The algorithm builds on results from [10, 11] concerning core solutions, retracts, and fact blocks. Roughly speaking, it starts by guessing a retract of the target instance of each data example. It then computes the fact blocks of the retracted target instances. Next, it guesses for each fact block a single source tuple that can intuitively be seen as being responsible for the creation of this fact block. Based on these guesses, a candidate LAV schema mapping is constructed, and finally, the algorithm tests in non-deterministic polynomial time if this LAV schema mapping fits the original data examples.

The lower bound is proved by a reduction from the NP-complete problem of whether two graphs are homomorphically equivalent. The lower bound holds even for a single example over a fixed schema, containing only two source tuples, or for two examples, each containing a single source tuple.

4. IMPLEMENTATION

We describe briefly our approach for the implementation of the GLAV fitting generation algorithm, presented in pseudocode in Figure 3. We will focus on the homomorphism extension test (also known henceforth simply as the *fitting test*), since constructing the canonical GLAV schema mapping is straightforward: it basically requires a linear pass over the set of data examples.

Let $\mathcal{E}$ be a set of data examples for the pair of schemas $(\mathbf{S}, \mathbf{T})$. Our system stores these data examples in the IBM DB2 database system. The logical design being used follows closely the schemas $(\mathbf{S}, \mathbf{T})$, with the addition of an extra attribute exid in each relation that identifies the data example to which a particular tuple belongs. To illustrate, reconsider the set of two data examples in step 3 of Figure 2, which we will denote by (I_1, J_1) and (I_2, J_2). These examples are stored in the database as in Figure 5.

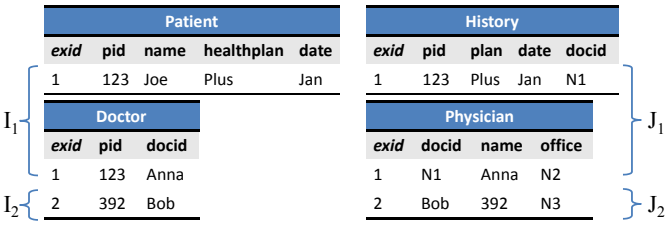

Figure 5: Database instance used to store the set of data examples in step 3 of Figure 2

The fitting test over $\mathcal{E}$ is implemented as a set of DB2 user-defined functions, one for each data example in $\mathcal{E}$. Intuitively, the function associated to an example $E_1 = (I_1, J_1) \in \mathcal{E}$ tries to find a witness to the failure of the fitting test: whether for some example $E_2 = (I_2, J_2) \in \mathcal{E}$, there exists a homomorphism from I_1 to I_2 which does not extend to a homomorphism from J_1 to J_2. In effect, if such a homomorphism exists, then there is no fitting GLAV schema mapping for the set $\mathcal{E}$ of data examples.

From the Chandra-Merlin theorem for conjunctive queries [8], we know that computing homomorphisms from an instance I_1 to an instance I_2 is equivalent to executing the canonical query of I_1 on I_2. The results of executing the canonical query of I_1 on I_2, hence the valuations for the variables in the canonical query of I_1, provide the homomorphisms from I_1 to I_2. Returning to the data examples above, the canonical query of I_1 has the following form:

$$Q^{I_1} : \text{Patient}(x, y, z, t) \wedge \text{Doctor}(x, u)$$

This query contains one relational atom for each tuple in I_1, and it employs repeated variables for multiple occurrences of a value in I_1 (e.g., the variable x for the value 123). Executing Q^{I_1} on I_1 returns the identity homomorphism from I_1 to itself. However, executing Q^{I_1} on I_2 returns an empty answer, since there are no homomorphisms from I_1 to I_2, as there are no Patient tuples in I_2.

In our implementation, the function associated to each example $E \in \mathcal{E}$ essentially executes a nested SQL query, where the outer

and inner blocks are (correlated) canonical queries of the source and target instances of E, respectively. More concretely, the nested SQL query has the following form:

```
select s1.exid from S1 s1, ..., Sp sp
where s2.exid = s1.exid and ... and sp.exid = s1.exid
      and C1 and ... and Ck
      and not exists
      ( select T1.exid from T1 t1, ..., Tq tq
        where t1.exid = s1.exid and ... and tq.exid = s1.exid
        and C'1 and ... and C'm )
```

In the query above, the from clauses contain one variable for each tuple of the source instance, and the target instance, respectively (in other words, one variable for each relational atom in the source and target canonical queries). The first set of equality conditions in the where clauses ensure that all tuples in the result belong to the same data example, identified by the same exid value. Furthermore, the conditions $C_i, 1 \leq i \leq k$ are equalities corresponding to multiple appearances of the same value in the source instance. Similarly, $C'_j, 1 \leq j \leq m$ correspond to multiple appearances of the same value in the target instance, or to a source value appearing in the target instance.

Continuing our illustration, the instantiation of the template above for the example (I_1, J_1) has the form shown at the end of this paragraph. It will return an empty result since there are no homomorphisms from I_1 to I_2. However, the corresponding query for (I_2, J_2) will return a non-empty result. The homomorphism $\{392 \rightarrow 123,\ \text{Bob} \rightarrow \text{Anna}\}$ from I_2 to I_1 cannot be consistently extended to a homomorphism from J_2 to J_1, hence it represents a witness to the failure of the fitting test.

```
select p.exid from Patient p, Doctor d
where d.exid = p.exid
      and p.pid = d.pid
      and not exists
      ( select h.exid from History h, Physician y
        where h.exid = p.exid and y.exid = p.exid
              and h.pid = p.pid and h.plan = p.healthplan
              and h.date = p.date and h.docid = y.docid
              and y.name = d.docid )
```

The fitting test succeeds if the functions corresponding to all the data examples return empty answers.

4.1 An Optimization

The number of tuple variables in the from clauses of the nested SQL queries described above is equal to the number of tuples in the source and target instances of the data examples. For arbitrary sets of data examples, this can lead to very large joins which translate to very long execution times for the fitting test. However, in our experience, such very large data examples proved to be uncommon in user-interactive schema mapping design scenarios.

The following theorem provides an optimization for the fitting test because it allows the test to be decomposed into tests that apply to disconnected components. By a *connected component* of a data example (I, J) we mean a set of elements from $adom(I) \cup adom(J)$ that forms a connected component of the Gaifman graph of (I, J) (i.e., the graph that contains an edge between two values if they co-occur in a fact of I or J).

THEOREM 4.1. *For all source instances I_1, I_2 and target instances J_1, J_2, the following are equivalent:*

- *Every homomorphism $h : I_1 \rightarrow I_2$ extends to a homomorphism $\widehat{h} : J_1 \rightarrow J_2$*
- *Either there is no homomorphism $h : I_1 \rightarrow I_2$, or for every connected component C of $I_1 \cup J_1$, every homomorphism $h : I_1 \cap C \rightarrow I_2$ extends to a homomorphism $\widehat{h} : J_1 \cap C \rightarrow J_2$.*

This result justifies the decomposition of the queries that implement the fitting test into simpler uncorrelated queries which can be evaluated independently. Intuitively, each simpler nested SQL statement executes the canonical queries of a data example subinstance corresponding to a connected component. As a consequence, the number of tuple variables in the from clauses decreases. Our experiments confirm the intuition that such decrease leads to an improvement in fitting test execution times.

As an important particular case, the fitting test function we generate becomes much simpler for a data example whose target instance contains only nulls. Intuitively, the nested SQL statement generated for such a data example is replaced by a sequence of two independent SQL statements without nesting, each implementing the canonical query of the source and the target instance, respectively. We will examine the effects of this optimization in the experimental analysis of Section 5.2.

5. EVALUATION

In the following, we present an experimental evaluation of our framework for designing schema mappings via data examples. We implemented our approach as a prototype in Java 6, with IBM DB2 Express-C v9.7 as the underlying database engine, running on a Dual Intel Xeon 3.4GHz Linux workstation with 4GB RAM.

We begin by giving an overview of the datasets used in our experiments. Then we present results concerning the efficiency of our implementation of the GLAV fitting algorithm, as well as some observations on the behavior of the fitting test when the structural characteristics of the underlying dataset change. Finally, we test our approach in a dynamic setting that simulates an interactive schema mapping refinement process.

The worst-case complexity of the GLAV fitting problem notwithstanding, the experimental results that we have obtained demonstrate the feasibility of interactively designing schema mappings using data examples. In particular, our experiments show that our system achieves very good performance in real-life scenarios.

5.1 Datasets

For our experimental evaluation, we use sets of data examples based on real-life schema mapping scenarios, as well as synthetically generated datasets.

Data examples based on real-life scenarios. We consider three real-life mapping scenarios. The first two involve scientific bibliographies (DBLP: `dblp.uni-trier.de/db` and Amalgam: `dblab.cs.toronto.edu/~miller/amalgam`, with mappings inspired by the Clio benchmarks[3]), and the third is from the biological domain (on fragments of the GUS:`www.gusdb.org` and BioSQL: `www.biosql.org` schemas). Some statistics on these scenarios are presented in the left half of Table 1. For instance, in the DBLP-Amalgam scenario, the source schema consists of 7 relations of average arity 6.5, and the target schema consists of 9 relations of the same average arity 6.5. Moreover, the schema mapping in this scenario is specified by 10 GLAV constraints, averaging 1.4 atoms in the left-hand-side and 2.2 atoms in the right-hand-side.

We use the *canonical sets of data examples*, introduced in Section 3.1, for each of the real-life mapping scenarios as input to our fitting algorithm. In the canonical set of data examples for a schema mapping, the source instances are the canonical instances of the

[3] queens.db.toronto.edu/project/clio/index.php#testschemas

	# of source relations	Avg. source arity	# of target relations	Avg. target arity	# of GLAV constraints	Avg. # of LHS atoms	Avg. # of RHS atoms	# of canonical examples	Time to generate canonical examples (s)	Avg. # of nonempty source relations	Avg. # of tuples per source relation	Avg. # of nonempty target relations	Avg. # of tuples per target relation
DBLP - Amalgam	7	6.5	9	6.5	10	1.4	2.2	10	4.8	1.4	1.0	2.2	1.1
Amalgam S1 - S2	15	6.7	27	2.0	71	1.2	2.1	15	9	1.9	1.0	10.7	1.1
GUS - BioSQL	7	6.4	6	5.5	8	1.6	1.9	7	2.3	1.6	1.1	2.1	2.3

Table 1: Statistics on real-life mapping scenarios and their canonical examples

left-hand-sides for each GLAV constraint in the mapping specification (pruned for isomorphic copies), and the target instances are the results of *chasing* each source instance with the schema mapping. We consider here a variant of the *chase procedure* (see [1]), called the *naive* chase. Given a source instance I, the naive chase produces a universal solution [9] for I, denoted by $chase_{\mathcal{M}}(I)$, as follows. For every GLAV constraint

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

in the specification of $\mathcal{M}$ and for every tuple $\mathbf{a}$ of source values from $\mathrm{adom}(I)$ such that $I \models \varphi(\mathbf{a})$, we add to $chase_{\mathcal{M}}(I)$ all facts in $\psi(\mathbf{a}, \mathbf{b})$, where $\mathbf{b}$ is a tuple of new nulls interpreting the existentially quantified variables $\mathbf{y}$.

Through the construction of the canonical sets of examples, the GLAV fitting algorithm will always return a positive answer on such sets of examples, since the chase procedure generates universal solutions. The canonical examples make a good test case to stress our implementation of the fitting test, because the guaranteed positive fitting result means the system indeed iterates over all source homomorphisms and checks for target extensions, since a failure witness is not encountered. In addition, by Theorem 3.7, the execution of the fitting algorithm on the set of canonical examples for a schema mapping $\mathcal{M}$ will produce a schema mapping that is logically equivalent to $\mathcal{M}$. Hence, the canonical examples are a natural choice as the starting point in an example driven refinement of an initial mapping specification.

Statistics on the canonical data examples of the real-life scenarios we used are presented in the right half of Table 1. For instance, the set of canonical data examples for the DBLP-Amalgam scenario consists of 10 data examples, and took 4.8 seconds to generate. Each canonical data example has on the average 1.4 nonempty source relations with one tuple each, and 2.2 nonempty target relations with 1.1 tuples each.

Synthetic data examples. To stress test the performance of the fitting test, we use synthetic sets of data examples for the schemas of the real-life mapping scenarios. The generation of the synthetic datasets is controlled via four parameters (n, t, d, f), as follows:

- n is the number of data examples in the dataset
- t is the number of tuples for each relation in the schemas
- d, called *diversity*, controls the number of occurrences of distinct values, hence determines the size of the domains for the source and target instances of a data example
- f, called *source value fraction*, determines how the domain of a target instance is split between source values and nulls.

More precisely, consider a pair of schemas $(\mathbf{S}, \mathbf{T})$. In our experiments, these schemas originate in the real-life mapping scenarios considered above. Given a parameter tuple (n, t, d, f), a set $\mathcal{E}$ of n data examples is generated as follows. Every example $E = (I, J) \in \mathcal{E}$ contains, for each $R \in (\mathbf{S}, \mathbf{T})$, an R relation consisting of t tuples. The domain of the source instance I is $D = \{a_1, \ldots, a_m\}$, where $m = d \times t \times \sum_{R \in \mathbf{S}} \mathrm{arity}(R)$. Each value in I is chosen through a uniform random pick from D. The domain of the target instance J is $D' = \{a_1, \ldots, a_k, b_{k+1}, \ldots, b_{m'}\}$, where $m' = d \times t \times \sum_{R \in \mathbf{T}} \mathrm{arity}(R)$, and $k = \min(|\mathrm{adom}(I)|, f \times m')$. The values $\{a_1, \ldots, a_k\}$ are among the source values appearing in I, while the rest of the values in D' are fresh nulls. Each value in J is picked through a uniform random choice from D'.

5.2 Fitting Test Efficiency

We now describe the experiments we conducted to determine the efficiency of the fitting test on the canonical sets of examples for the real-life scenarios. The results for this first set of experiments are presented in Table 2 (the rightmost column will be discussed in Section 5.3), and are averaged over 50 experimental runs.

We remark that performing the GLAV fitting test on the canonical examples for the real-life scenarios takes only a few seconds $(1.6, 3.6, 1.2)$, showing this is a promising first step in designing schema mapping via examples, following the workflow in Figure 1. Data examples of size comparable to the canonical examples are common in a user-interactive schema design environment.

	Number of examples	Size of each example (# of source + target tuples)	Initial fitting test (s)	Fitting test per user change (s)
DBLP - Amalgam	10	3.8	1.6	0.2
Amalgam S1 - S2	15	13.7	3.6	0.3
GUS - BioSQL	7	6.6	1.2	0.2

Table 2: Experimental results on canonical data examples of real-life scenarios

We conducted additional experimental analysis to determine the performance of the fitting test on different types of synthetic sets of data examples on the schemas of the real-life scenarios. The synthetic data examples are generated following the procedure in Section 5.1. We used sets of $n = 10$ data examples, where in each example, each source and each target relation was populated with $t = 3$ tuples. In the GUS-BioSQL scenario, this translates to 21 source tuples and 18 target tuples in each data example, for a total of 210 source tuples and 180 target tuples over the entire set of data examples. The experimental results for the synthetic data examples based on the GUS-BioSQL scenario are presented in Figure 6. The trends exhibited by the sets of synthetic data examples based on the remaining two scenarios, as well as for other values of the synthetic data generation parameters are not shown here, but are similar. The results we present are averaged over 200 experimental runs.

For the first experiment, in **Figure 6(a)**, we kept the source value fraction parameter f constant at 0.7, and we varied the diversity parameter d from 0.1 to 1.2 in increments of 0.1. The decrease in the fitting test execution times as the diversity increases is justified by the progressive simplification of the queries our system generates (see Section 4). When the diversity of an instance increases, repeated occurrences of a given value become less frequent, hence the canonical queries that are executed for the GLAV fitting test contain where clauses with fewer conditions, the rest being equal. This, combined with the fact that our implementation cannot make use of indexes (since we do not possess apriori knowledge on the

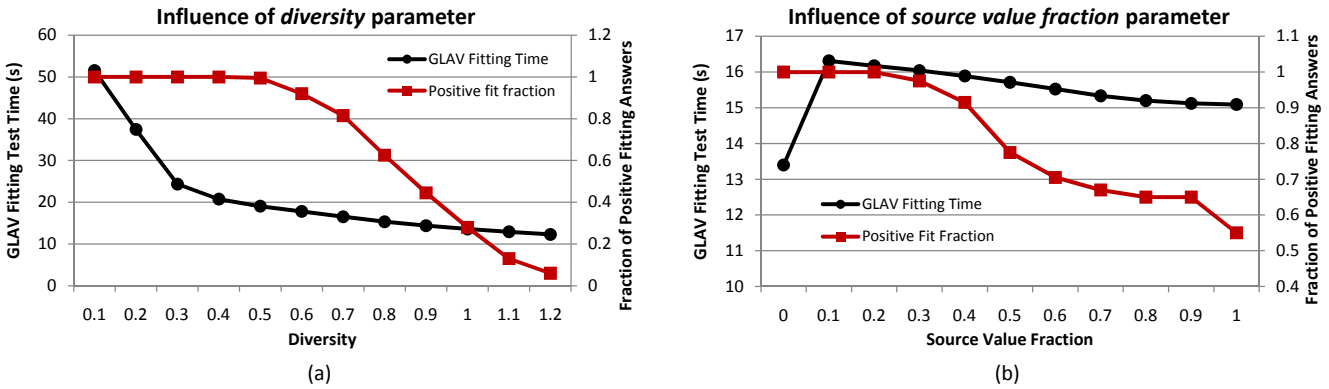

Figure 6: Experimental results for the fitting test on synthetic sets of data examples based on the GUS - BioSQL scenario: 10 data examples, each consisting of 39 tuples (21 in the source and 18 in the target) (a) Influence of the *diversity* parameter, when the *source value fraction* is 0.7; (b) Influence of the *source value fraction* parameter, when the *diversity* is 0.8.

values used in the data examples), explains the decrease in execution times. We also remark the decrease in the fraction of positive answers to the GLAV fitting test as the diversity parameter increases. Intuitively, the number of source homomorphisms increases as diversity increases (the canonical queries of the source instances become less constrained). Since the majority of values in the target instances are source values, it becomes increasingly difficult to find, for each source homomorphism h, a target homomorphism $\widehat{h}$ that is in fact an extension of h.

In the second experiment, we varied the source value fraction parameter between 0 and 1, while maintaining the diversity parameter constant at 0.8. As shown in **Figure 6(b)**, we observe an initial increase in the GLAV fitting time, followed by a decreasing trend. The increase occurs when the source value fraction increases above 0. For the source value fraction equal to 0, the target instances contain only nulls. As we mentioned in Section 4.1, the queries executed for the fitting test in this case can be simplified, leading to reduced execution times. When the source value fraction is greater than 0, the fitting test implementation executes the more involved nested and correlated SQL statements which take longer times to execute. However, as the source value fraction increases beyond 0.1, there is a slight decrease in the fitting test times. This happens because while the fraction of source values in the target instances increases, target homomorphisms become more pre-determined by the source homomorphisms. In other words, the space of target homomorphisms to be explored becomes smaller. In particular, when the source value fraction is 1, the target instances consist exclusively of source values, hence the fitting tests essentially perform only consistency checks. However, these are still more complex and require longer time than the simpler uncorrelated queries executed at source value fraction equal to 0. Finally, we remark the decrease in the fraction of positive answers to the fitting test as the source value fraction increases. This follows the intuition that as the source value fraction increases, the ratio of source values to nulls in the target instances increases, and thus it becomes more difficult to find consistent extensions of source homomorphisms to target homomorphisms.

Although the fitting test behaves well in terms of execution time on sets of data examples of practical size, the underlying high complexity of the decision procedure resurfaces in handcrafted scenarios like the following. Consider a source schema $\mathbf{S} = \{P\}$ and a target schema $\mathbf{T} = \{R\}$, where P and R are binary relations, and a set $\mathcal{E}$ consisting of the single data example (I, J), where $I = \{P(a, b_1), \ldots, P(a, b_n)\}$ and $J = \{R(c, b_1), \ldots, R(c, b_n)\}$. Even for relatively modest sizes of the data example above, such as for $n = 10$, the fitting test requires very long execution times. However, this is not entirely unexpected, since the fitting test was shown to be Π_2^{p}-complete (Theorem 3.8). In fact, in the scenario above, the number of source homomorphisms from I to itself is of the order n^n. We remark that encoding our fitting tests as quantified boolean formulas (QBF), followed by testing the satisfiability of such formulas using some of the leading QBF solvers[4] leads to similar very long execution times.

5.3 Interactive Mapping Refinement

To further investigate the practicality of our techniques, we simulate an interactive mapping refinement process via data examples, and study the behavior of the fitting test.

The simulation starts from a set of canonical examples for a real-life mapping scenario, or from a set of synthetic examples based on the schemas of a real-life scenario, and consists of a sequence of changes that a user might apply to the data examples to align them to the intended mapping semantics. The types of user changes we consider here are insertions and deletions of tuples in the target instances of data examples. In our experiments, each run consisted of a sequence of 20 changes, with equal probability tuple insertions or deletions. For a deletion change, the tuple to be deleted was chosen through a uniform random choice over the tuples in the target instances. For an insertion change, the target relation to be updated was chosen uniformly at random. Each value of the tuple to be inserted was chosen to be either a source value (with probability 0.4), or a null. If a null was inserted, it was either a fresh value (with probability 0.6), or one of the pre-existing nulls in the target instance. Furthermore, after the failure of a fitting test, the previous user change is undone with probability 0.75. This improves the realism of our simulation, by modeling situations in which the user decides to take back a change once the fitting test fails.

The results obtained following the simulations are presented in Table 2 for canonical sets of examples of the real-life scenarios, and in Table 3 for synthetic sets of examples based on the same scenarios. The synthetic examples were generated using the configuration tuple $(n = 10, t = 3, d = 0.6, f = 0.7)$, which was chosen here for illustration purposes, as the results obtained with other configuration parameters were similar. The numbers are averaged over 50 experimental runs. The times required for the fitting test scale as expected from the canonical data examples to the syn-

[4] http://www.qbflib.org/qbfeval

	Number of examples	Size of each example (# of source + target tuples)	Initial fitting test (s)	Fitting test per user change (s)
DBLP - Amalgam	10	48	17.7	1.8
Amalgam S1 - S2	10	126	222.4	23.1
GUS - BioSQL	10	39	14.2	1.5

Table 3: Interactive mapping refinement on synthetic sets of data examples ($n = 10, t = 3, d = 0.6, f = 0.7$) based on real-life scenarios

thetic data examples, given the significant increase in the size of each data example.

We remark a sharp decrease in the time needed for the fitting test after each of the user changes (column 5), from the initial test before the first user change (column 4). For instance, on the canonical sets of examples for the real-life scenarios, the decrease is from 1.6s to 0.2s, from 3.6s to 0.3s, and from 1.2s to 0.2s. This is a consequence of our implementation technique. In our experiments, the example sets were small enough to fit entirely in the DB2 buffer pool in main memory. Moreover, there were no other concurrent workloads on the database. Thus, the user changes updated data in main memory. Since there are no other workloads competing for buffer space, the updated examples are still in main memory when the next fitting test is performed. Hence these fitting tests execute much faster than the initial test, which needs to read the data from disk. In addition, after each user change on a data example E, only the definition of the function associated to E (see Section 4) has to be altered, redeployed to DB2 and recompiled.

Another consequence of our implementation approach is that a negative result for a fitting test is obtained faster than a positive result. In our experiments, positive fitting results represent a minority, which is not surprising, since intuitively it is difficult to maintain the existence of a fitting mapping through random changes. However, we observed that the times required for positive and negative fitting tests do not differ significantly. Hence, the sharp decrease in execution times during the interactive design simulation we discussed above is not significantly affected by this phenomenon.

6. CONCLUDING REMARKS

We developed a principled framework for the interactive design and refinement of schema mappings using data examples. A fundamental component of our framework is the GLAV fitting generation problem, which we brought into center stage and investigated in depth. We presented an algorithm that, given a finite set of data examples, either produces a GLAV schema mapping that fits these examples, or reports that none exists. We showed that the fitting schema mapping produced by the algorithm, if one such schema mapping exists, possesses several desirable properties. In particular, it is guaranteed to be the most general GLAV schema mapping that fits the given data examples, and also to have length linear in the total size of the input data examples. Furthermore, our GLAV fitting algorithm is complete for GLAV-schema-mapping design, in the sense that (up to logical equivalence) every GLAV schema mapping can be obtained by our algorithm when given the right set of data examples. In the worst-case, our GLAV fitting algorithm runs in exponential time, which is an inevitable consequence of the NP-hardness of the underlying GLAV fitting decision problem (in fact, we showed that this problem is complete for Π_2^{p}, hence harder than NP-complete). In spite of this worst-case complexity, extensive experimental evaluation of our prototype implementation established the viability of the example-driven approach to schema mapping design. In particular, interactive response times were achieved for real-life scenarios.

We believe that the example-driven approach to schema mapping design proposed here can be fruitfully combined with other extant approaches. Concretely, our approach may be used as a module within other schema-mapping-design systems, such as those described in the Introduction. After an initial candidate schema mapping is derived (for instance, on the basis of a visual specification), an initial set of data examples could be generated from this candidate schema mapping and presented to a user, who may either accept or modify the examples. By running our GLAV fitting generation algorithm on the updated set of data examples, a new schema mapping can then be generated, and this interactive process can be iterated until the user is satisfied with the result.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] B. Alexe, L. Chiticariu, R. J. Miller, and W. C. Tan. Muse: Mapping Understanding and deSign by Example. In *ICDE*, pages 10–19, 2008.

[3] B. Alexe, P. G. Kolaitis, and W. C. Tan. Characterizing schema mappings via data examples. In *ACM PODS*, pages 261–272, 2010.

[4] B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: Towards a Benchmark for Mapping Systems. *PVLDB*, 1(1):230–244, 2008.

[5] P. Barceló. Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58, 2009.

[6] P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing Mapping Composition. *VLDB Journal*, 17(2):333–353, 2008.

[7] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *VLDB*, pages 1267–1270, 2005.

[8] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[9] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.

[10] R. Fagin, P. G. Kolaitis, A. Nash, and L. Popa. Towards a Theory of Schema-Mapping Optimization. In *ACM PODS*, pages 33–42, 2008.

[11] R. Fagin, P. G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *ACM TODS*, 30(1):174–210, 2005.

[12] G. H. Fletcher and C. M. Wyss. Towards a general framework for effective solutions to the data mapping problem. *Journal on Data Semantics*, XIV, 2009.

[13] G. H. L. Fletcher, M. Gyssens, J. Paredaens, and D. V. Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *TKDE*, 21(6):939–942, 2009.

[14] G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *JACM*, 57(2), 2010.

[15] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *ACM SIGMOD*, pages 805–810, 2005.

[16] P. G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *ACM PODS*, pages 61–75, 2005.

[17] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM PODS*, pages 233–246, 2002.

[18] H. Mannila and K.-J. Räihä. Automatic generation of test data for relational queries. *JCSS*, 38(2):240–258, 1989.

[19] C. Olston, S. Chopra, and U. Srivastava. Generating example data for dataflow programs. In *ACM SIGMOD*, pages 245–256, 2009.

[20] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[21] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *ICDT*, pages 89–103, 2010.

[22] B. ten Cate, P. G. Kolaitis, and W. C. Tan. Database constraints and homomorphism dualities. In *CP*, 2010.

[23] L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. In *ACM SIGMOD*, pages 485–496, 2001.