ROMA TRE
UNIVERSITÀ DEGLI STUDI

Dipartimento di Ingegneria
Via della Vasca Navale, 79
00146 Roma, Italy

Reuse of Schema Mappings for Data Transformation Design

Paolo Atzeni[1], Luigi Bellomarini[1,2], Paolo Papotti[3], Riccardo Torlone [1]

RT-DIA-217-2017 **February 2017**

(1) Università Roma Tre

(2) Banca d'Italia

(3) Arizona State University

ABSTRACT

The definition of data transformations between heterogeneous schemas is a critical activity of any database application. Currently, automated tools provide high level interfaces for the discovery of correspondences between elements of schemas, but transformations (i.e., schema mappings) need to be manually specified every time from scratch, even if the problem at hand is similar to one that has already been addressed. Schema mappings are precisely defined over a pair of schemas and cannot directly be reused on different scenarios. We tackle this challenge by introducing a framework to generalize schema mappings as *meta-mappings*: formalisms that describe transformations between generic data structures called *meta-schemas*. Meta-mappings describe, in abstract terms, database transformations and are suitable to capture enterprise knowledge from previously defined schema mappings. We present novel techniques to infer meta-mappings from schema mappings, to organize them into a searchable repository, and to leverage the repository to propose to the users data transformations suitable for their needs. We study effectiveness and efficiency in an extensive evaluation over both real-world and synthetic transformation scenarios, and show that our system can infer, store, and search millions of meta-mappings in seconds.

1 Introduction

In large institutions, there is often the need to transform data between different schemas to reconcile heterogeneities, to satisfy novel requirements, or to fulfill external specifications. Although this problem has been largely studied in the database literature since its beginning [20], it is still a time-consuming and error-prone task in practice.

While several tools already support data architects in the creation of data transformations from scratch [9], we observe that data transformation scenarios are often similar to one another and so reuse of existing solutions could reduce the human effort in the design step. Given the overwhelming amount of "enterprise knowledge" stored in traditional data warehousing and in data-lakes [10, 13], it is recognized that reuse is a challenge of increasing importance [1].

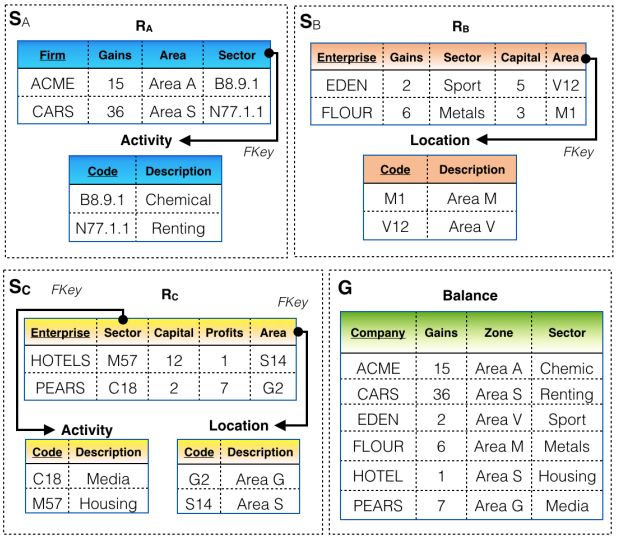

Figure 1: The business register at a central bank, **G**, and three data providers.

Unfortunately, there is no obvious approach to transformation reuse. Consider the following example.

Example 1. *A central bank maintains a register with balance data from all companies in the country (Figure 1). This register has schema* **G**, *with relation* `Balance` *storing for each company, its gains, zone of operation, and economic sector. External providers send data to the bank in different forms. Provider A adopts a schema* $\mathbf{S}_A$, *with a relation* R_A *for companies (firms), with gains, zone of operation, here named* `Area`, *and economic sector, whose code in* R_A *refers to relation* `Activity` *with a description. Provider B adopts a schema* $\mathbf{S}_B$, *with a relation* R_B *for companies (enterprises), their gains, sector, capital, and area, whose code in* R_B *refers to relation* `Location` *with a description. Data can be moved from* $\mathbf{S}_A$ *and* $\mathbf{S}_B$ *into* **G**, *by using schema mappings* M_A *and* M_B *described by the following formulas:*

σ_A: $R_A(f, g, a, s), Activity(s, d) \rightarrow Balance(f, g, a, d)$.

σ_B: $R_B(e, g, s, c, a), Location(a, d) \rightarrow Balance(e, g, d, s)$.

The example shows a *transformation scenario*: there is a family of transformations, which share the goal of importing data from various providers into **G**. Two transfor-

mations are implemented by mappings M_A and M_B, whose differences are due to the differences between $\mathbf{S}_A$ and $\mathbf{S}_B$.

This scenario gets more meaningful at scale, as there can be many other data providers (such as $\mathbf{S}_C$ in the Figure), each with its specific customization to the source schema. This requires the manual design of several ad-hoc mappings. The reuse of previously defined schema mappings, such as M_A and M_B, would save the definition of a new transformation for other schemas, such as $\mathbf{S}_C$. Unfortunately, a mapping cannot directly be applied on a different schema, even in presence of only minor differences between the schemas. Schema mappings are widespread because they precisely characterize a transformation at a level of detail that enables both general reasoning and efficient execution. A mapping captures the semantics of a transformation for a given pair of schemas, and even a simple variation in a relation, such as a different name or a different number of attributes, makes it not applicable. To be reusable, a mapping should be independent of its specifities, but, at the same time, harness the essence of the transformation to work for similar schemas.

In this paper, we present a solution for mapping reuse. At the basis, there is a meta-level approach, which allows the description of transformations in a generic way. In practice, we adopt a data dictionary to "describe" schemas independently of the names of the relations and attributes. Our dictionary includes the specification of constraints (keys and foreign keys), which we leverage to better characterize the transformations and sort out specific ambiguities. The basic idea is that some of the attribute names can be represented by variables and that conditions on them can be specified by means of participation into constraints. However, this is not sufficient in general and it is not easy to achieve generalization while guaranteeing reusability of transformations, as we elaborate in the next examples.

Example 2. *A "generic" mapping Σ_A, obtained from M_A by simply making it independent of the names of the relations and attributes (used as variables in rules), could be informally described as follows:*

Σ_A*:* for each *relation* r with *key* f *and attributes* g, a, s
for each *relation* r' with *key* s *and any attribute* d
with a fk constraint from s *of* r *to* s *of* r'
there exists *a relation* r'' with *key* f *and attributes* g, a, d.

The generic mapping Σ_A properly maps $\mathbf{S}_A$ to $\mathbf{G}$. It seems a valid compromise between precision, a transformation that expresses the semantics of the original mappings, and generality, as it can be applicable over different schemas. However, Σ_A fails the latter requirement, as it is not applicable on $\mathbf{S}_B$. Since there are no constraints on g and a, they would be bound to any of `Gains`, `Sector` and `Capital`, eventually copying `Capital` into the target, which is not desired.

Example 3. *If we followed the same approach starting from $\mathbf{S}_B$, then we would obtain:*

Σ_B*:* for each *relation* r with *key* e *and attributes* g, s, c, a
for each *relation* r' with *key* a *and any attribute* d
with a fk constraint from a *of* r *to* a *of* r'
there exists *a relation* r'' with *key* e *and attributes* g, d, s.

The generic mapping Σ_B does not even apply to $\mathbf{S}_B$. As we have no restrictions w.r.t. what attributes are represented by g, s and c, variables g and s eventually bind to `Capital`, wrongly copying it into the target, in contrast to the original semantics of M_B.

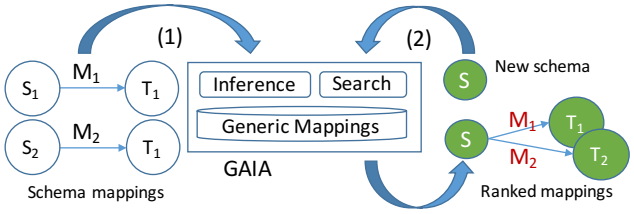

Figure 2: The architecture of GAIA.

The above difficulties suggest the use of explicit references to some attributes, as in the following example.

Example 4. *Consider the generic mapping:*

Σ'_B*:* for each *relation* r with *key* e *and attributes* g, s, c, a
for each *relation* r' with *key* a *and any attribute* d
with a fk constraint from a *of* r *to* a *of* r'
where $g = Gains,\ s = Sector,\ c = Capital$
there exists *a relation* r'' with *key* e *and attributes* g, d, s.

It correctly applies to $\mathbf{S}_B$ *as the ambiguous variables are constrained to specific values. However, it does not work for* S_A*, since attribute* `Capital` *is not present, and* Σ'_B *does not expect* `Sector` *to be involved in a foreign key. It is possible to choose other combinations of constraints, which make* Σ'_B *effective for both the schemas and therefore reusable:*

$$g = Gains,\ s \neq Gains,\ s \neq Capital,\ c \neq Gains,\ c \neq Sector$$

With the above constraints, g *can bind only to* `Gains` *of* R_A*, while* s *and* c *only to* `Area`.

The examples show that there is a combination of attributes, identified by their names and role as constraints, that form a correct and useful generic mapping. Once pinpointed, a generic mapping can be stored in a repository, so that it is possible to retrieve and adapt it, for example when a new source must be mapped to a target covered by a previous transformation. In our example, if also $\mathbf{S}_C$ (Figure 1) needs to be mapped to $\mathbf{G}$, the generic mapping in Example 4 can be retrieved from the repository, and a user only needs to change one constant ("Profits" to "Gains") before immediately applying it.

Unfortunately, two main challenges make reuse hard. First, the set of attribute conditions that defines the semantics of the transformation is unknown and there are exponential possible combinations for a single schema mapping. Second, given a new schema, there can be an overwhelming number of generic mappings that can be applied.

In this work, we tackle these two challenges and introduce an end-to-end system, GAIA, for mapping reuse. GAIA supports two main tasks, as depicted in Figure 2: (1) infer a generic mapping from one (or many) input schema mappings, (2) return suitable transformations from the repository. Our general framework enables the retrieval of transformations for different applications. We consider three main use cases: (a) given a source and a target schema, provide a ranked list of possible transformations that move the data from a source to a target database (as in the previous examples); (b) given a source schema, generate possible target schemas along with a ranked list of possible transformations (in the Figure); (c) given a target schema, propose reasonable sources and a ranked list of possible transformations.

We support these tasks with the following contributions:

1. A semantics to precisely characterize when an abstract notion of schema mapping, which we call *meta-mapping*, "fits" a transformation scenario (Section 3). This solution enables the reuse of schema mappings.
2. The problem of finding meta-mappings that fit a scenario is not trivial, as not all the requirements can be satisfied at the same time. We formally characterize the problem and propose an algorithm to *infer* meta-mappings from schema mappings. The algorithm is based on the main theoretical result of this paper, a necessary and sufficient condition for the fitness of a meta-mapping, used in a constructive way to generate fitting meta-mappings with an approach that also extends previous efforts for the definition of schema mappings by example (Section 4).
3. Given a repository of generic mappings, we want to be able to find the most suitable one and to instantiate it to be used for the specific problem. Since for a given scenario multiple possible matches are possible, we introduce an algorithm for the *retrieval* of transformations for the scenario at hand (Section 5). The algorithm is parametric w.r.t. the scoring mechanism, and we define our feature-based measure, the *coverage*, to quantify how suitable a meta-mapping is for a given setting. We use this measure to rank alternative meta-mappings as well as to index them to speed up the retrieval process.

The algorithms have been implemented and experimentally evaluated on a scenario with more than 20,000 real-world data transformations over 40,000 schemas (Section 6). We show that our solution is effective in practice and the optimizations enable an interactive use of the system. Proofs and additional experiments are in the Appendix.

2 Background

In this section we recall the preliminary notions of schema mapping [8, 16] and meta-mapping, which originates from the notion of *template mappings* proposed in [18]. The former will be used to model specific data transformations, and the latter will be used to model *generic* transformations between schemas. While we adopt template mappings as the syntax for our approach, our contributions are orthogonal to this choice and apply on any mapping formalism with variables over attribute names (e.g., [14]).

2.1 Schema mappings

Let $\mathbf{S}$ (the source) and $\mathbf{T}$ (the target) be two relational schemas and let $Inst(\mathbf{S})$ and $Inst(\mathbf{T})$ denote the set of all possible instances of $\mathbf{S}$ and $\mathbf{T}$ respectively. A *(schema) mapping* M for $\mathbf{S}$ and $\mathbf{T}$ is a binary relation over their instances, that is, $M \subseteq Inst(\mathbf{S}) \times Inst(\mathbf{T})$ [6].

We consider mappings expressed by a single *source-to-target tuple-generating-dependency* (st-tgd) σ of the form: $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$ where $\mathbf{x}$ and $\mathbf{y}$ are two disjoint sets of variables, $\phi(\mathbf{x})$ (the left-hand-side, LHS) is a conjunction of atoms involving relations in $\mathbf{S}$ and $\psi(\mathbf{x}, \mathbf{y})$ (the right-hand-side, RHS) is a conjunction of atoms involving relations in $\mathbf{T}$. The dependency represents a mapping M in the sense that $(I, J) \in M$ if and only if (I, J) satisfies σ. Given I and σ, we can compute a suitable J in polynomial time by applying the *chase procedure* to I using σ [8]: the result is called the *universal solution* of I under σ. A universal solution may have labeled nulls denoting unknown values.

Example 5. *Let us consider the schemas* $\mathbf{S}_A$, $\mathbf{S}_B$ *and* $\mathbf{G}$ *of Figure 1 that we recall next for convenience.*

$$\mathbf{S}_A = \{R_A(\underline{Firm}, Gains, Area, Sector), \\ Activity(\underline{Code}, Description)\}$$

$$\mathbf{S}_B = \{R_B(\underline{Enterprise}, Gains, Sector, Capital, Area), \\ Location(\underline{Code}, Description)\}$$

$$\mathbf{G} = \{Balance(\underline{Company}, Gains, Zone, Sector)\}$$

A possible mapping between $\mathbf{S}_A$ *and* $\mathbf{G}$ *is the st-tgd*[1]

$$\sigma_A : R_A(f, g, a, s), Activity(s, d) \rightarrow Balance(f, g, a, d).$$

Intuitively, the application of the chase procedure to the instance of $\mathbf{S}_A$ *in Figure 1 using* σ_A *tries to enforce this dependency by generating one tuple in the target for each pair of tuples in the source for which there is a binding to the LHS of the dependency. The result consists in the first two tuples in relation* `Balance` *in Figure 1.*

A possible mapping between $\mathbf{S}_B$ *and* $\mathbf{G}$ *is the s-t tgd*

$$\sigma_B : R_B(e, g, s, c, a), Location(a, d) \rightarrow Balance(e, g, d, s).$$

The result of the chase consists in the third and in the fourth tuple of relation `Balance`.

Specifying a mapping through st-tgd's is a consolidated practice in data transformation and integration, as they offer a straightforward mechanism to define a desired relationship between the source and the target schema. Since we refer to schema mappings expressed by one dependency only, we will blur the distinction between the mapping and the dependency and simply refer to "the mapping".

2.2 Meta-mappings

Basically, a *meta-mapping* is a mapping over the catalog of a relational database and describes generic mappings between relational schemas [18]. Specifically, in a meta-mapping, source and target are both defined over the following schema, called *(relational) dictionary*: Rel($\underline{name}$), Att($\underline{name}, in$), Key($\underline{name}, in$), FKey($\underline{name}, in, refer$) (note that, for the sake of simplicity, we consider here a simplified version of the relational model). An instance S of the dictionary is called *m-schema* and describes relations, attributes and constraints of a (standard) relational schema $\mathbf{S}$.

Example 6. *The m-schemas* $\mathsf{S_A}$ *and* G *for the schemas* $\mathbf{S}_A$ *and* $\mathbf{G}$ *in Example 5 are as follows:*

$\mathsf{S_A}$

Rel
name
R_A
Activity

Key		
name	in	
Firm	R_A	
Code	*Activity*	

Att	
name	in
Gains	R_A
Area	R_A
Descrip.	*Activity*

FKey		
name	in	refer
Sector	R_A	*Activity*

G

Rel
name
Balance

Key		
name	in	
Company	*Balance*	

Att	
name	in
Gains	*Balance*
Zone	*Balance*
Sector	*Balance*

[1]We omit quantifiers for the sake of readability

We assume, thereinafter, that given a schema, its corresponding m-schema is also given, and vice versa.

A meta-mapping is expressed by means of an st-tgd over dictionaries that describes how a source m-schema is transformed into a target m-schema.

Example 7. *Mapping σ_A of Example 5 can be expressed, in generic terms, by the following meta-mapping:*

$$\Sigma_A : \mathsf{Rel}(R), \mathsf{Key}(K_1, R), \mathsf{Att}(A_1, R), \mathsf{Att}(A_2, R),$$
$$\mathsf{FKey}(F, R, S), \mathsf{Rel}(S), \mathsf{Key}(K_2, S), \mathsf{Att}(A_3, S) \rightarrow$$
$$\mathsf{Rel}(T), \mathsf{Key}(K_1, T), \mathsf{Att}(A_1, T), \mathsf{Att}(A_2, T), \mathsf{Att}(A_3, T).$$

This st-tgd can be expressed in a graphical way as illustrated in Figure 3 and describes a general transformation that takes two source relations R and S (whichever they be) linked by a foreign key F and generates a target relation T obtained by joining R and S and taking the key K_1 and the attributes A_1 and A_2 from relation R and the attribute A_3 from S.

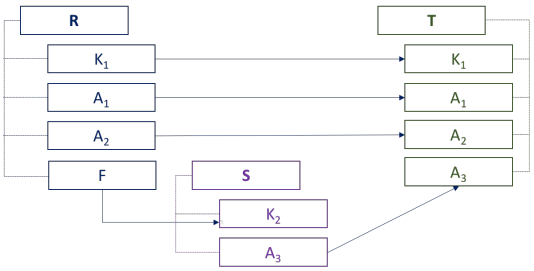

Figure 3: Graph representation of a meta-mapping.

There are indeed possible subtleties that could arise from the application of the chase in presence of existential quantifications in meta-mappings producing duplications of relations in the result. This is avoided by assuming that, for each existentially quantified variable, there is a target constraint, known as *egd* [8], ensuring that whenever two relations in the target have the same structure, then they coincide.

Example 8. *The application of the chase to the m-schema $\mathsf{S_A}$ in Example 6 using Σ_A of Example 7 generates the following target m-schema where $\perp_R$ is a null denoting a placeholder for a relation name.*

Rel

name
$\perp_R$

Key

name	in
Firm	$\perp_R$

Att

name	in
Gains	$\perp_R$
Area	$\perp_R$
Description	$\perp_R$

Note that a meta-mapping operates at schema level rather than at data level and thus provides a means for describing generic transformations.

It is finally possible to generate a schema mapping from a meta-mapping for a specific pair of source and target schemas with a special procedure in [18]. For example, this procedure is able to reconstruct the schema mapping σ_A of Example 5 from the meta-mapping in Example 7 and the schemas $\mathbf{S}_A$ and $\mathbf{G}$ of Figure 1. The schema mapping we obtain can be then executed with known techniques, for example with the chase procedure, or translating it into SQL.

3 From mappings to meta-mappings

In this section we describe how a schema mapping M can be suitably generalized into a meta-mapping $\mathscr{M}$, making it independent of the specificities of its source and target schemas. We introduce *canonical meta-mappings*, directly generated from m-schemas, and show that they can be the basis to build "correct" meta-mappings.

3.1 Canonical meta-mappings

An important property of a meta-mapping is the ability to "capture" a mapping, as formalized by property of *fitness*.

We first make explicit the correspondences between attributes expressed by a mapping. For this a preliminary notion is needed. Let σ be a mapping from a source schema $\mathbf{S}$ to a target schema $\mathbf{T}$ and let S and T be the m-schemas of $\mathbf{S}$ and $\mathbf{T}$, respectively. The *projection* of σ on S and T is a pair of m-schemas, denoted by S^σ and T^σ, obtained from S and T as follows: (i) for each variable x that occurs in both the LHS and the RHS of σ, we equate, if they are different, all the attributes in S and T on which x occurs using the same distinct constant, and (ii) for each remaining constant c that occurs in T but not in S, we replace all the occurrences of c with the same distinct labelled null.

Example 9. *The projection of the mapping σ_A in Example 5 to the m-schemas in Example 6 is as follows.*

$\mathsf{S}_\mathsf{A}^{\sigma_\mathsf{A}}$

Rel	
name	
R_A	
Activity	

Key		
name	in	
$\boldsymbol{c}_f$	R_A	
Code	*Activity*	

Att	
name	in
Gains	R_A
$\boldsymbol{c}_a$	R_A
$\boldsymbol{c}_d$	*Activity*

FKey		
name	in	refer
Sector	R_A	*Activity*

$\mathsf{G}^{\sigma_\mathsf{A}}$

Rel	
name	
$\perp_B$	

Key		
name	in	
$\boldsymbol{c}_f$	$\perp_B$	

Att	
name	in
Gains	$\perp_B$
$\boldsymbol{c}_a$	$\perp_B$
$\boldsymbol{c}_d$	$\perp_B$

We are ready to introduce the notion of fitness.

Definition 1. *A meta-mapping Σ* fits *a mapping σ from a source schema $\mathbf{S}$ to a target schema $\mathbf{T}$ if T^σ is the universal solution of S^σ under Σ.*

For a fitting meta-mapping Σ, we also say that the pair $(\mathsf{S}^\sigma, \mathsf{T}^\sigma)$ is a *(universal) example* for the meta-mapping Σ. This definition can be easily extended to a multiplicity of mappings that refers to a common domain of interest: we say that a meta-mapping Σ *fits a transformation scenario* $\mathscr{H} = \{M_1, \ldots, M_n\}$ if it fits each mapping $M_i \in \mathscr{H}$.

In the following, given a tuple $(a_1, \ldots, a_k)$ in the instance I of a schema $R(A_1, \ldots, A_k)$, we call the atom $R(a_1, \ldots, a_k)$ a *fact* of I.

Definition 2. *The* canonical meta-mapping Σ *for a mapping σ from a source schema $\mathbf{S}$ to a target schema $\mathbf{T}$ contains an st-tgd $\forall \mathbf{x}(q_S(\mathbf{x}) \rightarrow \exists \mathbf{y} q_T(\mathbf{x}, \mathbf{y}))$, where: (i) $q_S(\mathbf{x})$ is the conjunction of all the facts in S^σ, with each value in S^σ replaced by a universally quantified variable in $\mathbf{x}$ and (ii) $q_T(\mathbf{x}, \mathbf{y})$ is the conjunction of all the facts of T^σ with each value in T^σ not occuring in S replaced with an existentially quantified variable in $\mathbf{y}$.*

Example 10. *Consider again the mapping σ_A in Example 5. The projections of σ_A on its source and target m-schemas are presented in Example 9. From them, we obtain precisely the canonical-meta mapping Σ_A discussed in Example 7. It can be easily verified that Σ_A fits σ_A since by chasing $\mathsf{S}_\mathsf{A}^{\sigma_\mathsf{A}}$ using Σ_A we obtain $\mathsf{G}^{\sigma_\mathsf{A}}$ (both in Example 9).*

For a given pair of m-schemas, the canonical meta-mapping always exists and can be determined in linear time by applying Definition 2 in a constructive way. It can be observed, that our notion extends that of *canonical GLAV schema mappings* [3], defined at data level.

Unfortunately, a canonical meta-mapping does not necessarily fit the example it originates from.

Example 11. *Let us consider the mapping σ_B in Example 5 that we recall next for convenience.*

$$\sigma_B : R_B(e, g, s, c, a), L(a, d) \rightarrow B(e, g, d, s).$$

The projection of σ_B to S_B (the source) and G (the target) are the following m-schemas.

$\mathsf{S}_\mathsf{B}^{\sigma_\mathsf{B}}$

Rel
name
R_B
Location

Key		
name		in
$\boldsymbol{c}_e$		R_B
Code	*Location*	

Att	
name	in
Gains	R_B
Sector	R_B
Capital	R_B
$\boldsymbol{c}_d$	*Location*

FKey		
name	in	refer
Area	R_B	*Location*

$\mathsf{G}^{\sigma_\mathsf{B}}$

Rel
name
$\perp_B$

Key		
name		in
$\boldsymbol{c}_e$		$\perp_B$

Att	
name	in
Gains	$\perp_B$
Sector	$\perp_B$
$\boldsymbol{c}_d$	$\perp_B$

The canonical meta-mapping Σ_B for σ_B is then as follows.

$$\Sigma_B : \mathsf{Rel}(R), \mathsf{Key}(K_1, R), \mathsf{Att}(A_1, R), \mathsf{Att}(A_2, R), \mathsf{Att}(A_3, R),$$
$$\mathsf{FKey}(F, R, S), \mathsf{Rel}(S), \mathsf{Key}(K_2, S), \mathsf{Att}(A_4, S) \rightarrow$$
$$\mathsf{Rel}(T), \mathsf{Key}(K_1, T), \mathsf{Att}(A_1, T), \mathsf{Att}(A_2, T), \mathsf{Att}(A_4, T).$$

This meta-mapping does not fit σ_B since the chase of $\mathsf{S}_\mathsf{B}^{\sigma_\mathsf{B}}$ using Σ_B is an m-schema that includes all the tuples in $\mathsf{G}^{\sigma_\mathsf{B}}$ plus the tuple $\langle Capital, \perp_B \rangle$ in Att.

The example above shows that canonical mappings may fail to identify structural ambiguities, such as, in our example, the multiple non-key attributes (A_1, A_2, A_3) related to the same relation (R), which can bind to any of the attributes *Gains*, *Sector* and *Capital*, eventually copying *Capital* into the target, which is not desired.

We formalize this intuition by introducing the notion of *(potentially) ambiguous variables* and studying its connections to the fitness of a meta-mapping.

As a preliminary notion, we say that the *position* $\tau(q(\mathbf{x}), x)$ of a variable x in a conjunctive formula $q(\mathbf{x})$ is the set of pairs $\{(a_1, p_1), \ldots, (a_n, p_n)\}$, where a_i is the predicate of an atom in $q(\mathbf{x})$ containing x, and p_i is the relative position of x in a_i. For example, given: $q(\mathbf{x}) : \mathsf{Rel}(R), \mathsf{Key}(K, R), \mathsf{Att}(A, R)$ we have: $\tau(q(\mathbf{x}), R) = \{(\mathsf{Rel}, 1), (\mathsf{Key}, 2), (\mathsf{Att}, 2)\}$.

Definition 3. *Two distinct variables x_i, x_j are* potentially ambiguous *in a conjunctive formula $q(\mathbf{x})$, if $\tau(q(\mathbf{x}), x_i) \cap \tau(q(\mathbf{x}), x_j) \neq \emptyset$.*

The above notion (which can be generalized to the case of multiple variables by considering them pairwise) applied to the premise q_S of a canonical meta-mapping, captures the "structural ambiguity" of the two variables, that is, their presence in the same position in homonym atoms. However, additional facts are generated in the target only if the ambiguity is actually "activated" by a homomorphism in some chase step, as captured by the next definition.

We first recall that: (i) a *homomorphism* from a conjunctive formula $q(\mathbf{x})$ to an instance I is a mapping of the variables $\mathbf{x}$ into constants and variables, such that for every atom $P(x_1, \ldots, x_n)$ of $q(\mathbf{x})$, the fact $P(h(x_1), \ldots, h(x_n))$ is in I and that (ii) a homomorphism h from $q(\mathbf{x})$ to I *extends to* another homomorphism h' from $q'(\mathbf{y})$ to J, if for every variable x in the intersection of $\mathbf{x}$ and $\mathbf{y}$, $h(x) = h'(x)$.

Definition 4. *Two distinct variables x_i, x_j are* ambiguous *in a canonical meta-mapping $\Sigma : q_S \rightarrow q_T$ if: (1) they are potentially ambiguous in q_S, (2) there exists a homomorphism h that maps some atom α whose predicate occurs in $\tau(q_S, x_i) \cap \tau(q_S, x_j)$, into the same fact, and (3) one of the following holds: (a) x_i and x_j are not potentially ambiguous in q_T; (b) h extends to a homomorphism h' that maps α into different facts.*

Example 12. *In Example 11, variables A_1, A_2, A_3 of Σ_B are ambiguous because: (i) they are pairwise potentially ambiguous in q_S ($\tau(q_S, A_1) \cap \tau(q_S, A_2) \cap \tau(q_S, A_3) = \{(\mathsf{Att}, 1)\}$); (ii) they are not pairwise potentially ambiguous in q_T (A_3 does not appear in the conclusion). By contrast, variables A_1, A_2 of Σ_A in Example 7 are not ambiguous since they are potentially ambiguous in both q_S and q_T.*

Condition (b) of Definition 4 covers the subtle cases in which x_i and x_j are potentially ambiguous in both q_S and q_T (hence condition (a) does not hold), yet the atoms of the target in which x_i and x_j appear contain terms bound to different values by an extended homomorphism.

We are now ready to give an effective characterization of fitness for a meta-mapping, whose proof is omitted because of space limitations and can be found in Appendix A.

Theorem 1. *A canonical meta-mapping Σ fits its mapping σ if and only if it does not contain ambiguous variables.*

3.2 Fitting meta-mappings

Let us now come to the problem of finding a fitting meta-mapping, given a non-fitting canonical meta-mapping.

Taking inspiration from Theorem 1, which provides a necessary and sufficient condition to guarantee fitness, we introduce, in canonical meta-mappings, constraints that avoid the presence of ambiguous variables. Specifically, given a canonical meta-mapping $\forall \mathbf{x}(q_S(\mathbf{x}) \rightarrow \exists \mathbf{y} q_T(\mathbf{x}, \mathbf{y}))$, we consider meta-mappings of the form:

$$\forall \mathbf{x}(q_S(\mathbf{x}), \gamma(\mathbf{x}) \rightarrow \exists \mathbf{y} q_T(\mathbf{x}, \mathbf{y}))$$

where $\gamma(\mathbf{x})$ is a conjunction of constraints of the form: (i) $x_i = c_i$, (ii) $x_i \neq c_i$, or (iii) $x_i < x_j$, in which x_i and x_j are variables and c_i is a constant. Indeed, we assume, as pretty natural in all applications, that data domains are ordered.

We consider in particular two main classes of mappings: (i) mappings involving equality and inequality constraints, which we call *explicit mappings* (*positive*, *negative* or *hybrid* if they involve equalities only, inequalities only or both, respectively); (ii) mappings involving constraints of the form $x_i < x_j$, which we call *laconic mappings*.

Example 13. *Let $q_S(\mathbf{x})$ and $q_T(\mathbf{x}, \mathbf{y})$ be the LHS and the RHS of the meta-mapping Σ_B in Example 11, respectively. Possible extensions of Σ_B with constraints are reported below. They belong to different classes: positive (Σ_B^P), negative (Σ_B^N), hybrid (Σ_B^H), and laconic (Σ_B^L). All of them fit σ_B.*

$$\Sigma_B^P : q_S(\mathbf{x}), A_1 = \mathit{Gains}, A_2 = \mathit{Sector}, A_3 = \mathit{Capital} \rightarrow q_T(\mathbf{x}, \mathbf{y})$$

$$\Sigma_B^N : q_S(\mathbf{x}), A_1 \neq \mathit{Sector}, A_1 \neq \mathit{Capital}, A_2 \neq \mathit{Gains}, A_2 \neq \mathit{Capital}, A_3 \neq \mathit{Gains}, A_3 \neq \mathit{Sector} \rightarrow q_T(\mathbf{x}, \mathbf{y})$$

$$\Sigma_B^H : q_S(\mathbf{x}), A_1 = \mathit{Gains}, A_2 \neq \mathit{Gains}, A_2 \neq \mathit{Capital}, A_3 \neq \mathit{Gains}, A_3 \neq \mathit{Sector} \rightarrow q_T(\mathbf{x}, \mathbf{y})$$

$$\Sigma_B^L : q_S(\mathbf{x}), A_3 < A_1 < A_2 \rightarrow q_T(\mathbf{x}, \mathbf{y})$$

Explicit mappings and laconic mappings are relevant for different reasons.

- The presence of constants in explicit mappings guarantees a form of semantic compliance of the meta-mapping with the scenario at hand. For instance, the meta-mappings in Example 13 are likely to be more suitable in application domains involving `Gains` and `Capital` rather than those involving `Students` and `Courses`. On the other hand, while negative mappings cover more cases but tend to be too general, positive mappings are more restrictive but possibly more suitable to capture a specific application domain. Therefore, to exploit the advantages of both, it is useful to produce also hybrid mappings, which involve both equality and inequality constraints.

- Laconic mappings use the strict total order $<$ on the domains of the attribute names to solve ambiguities. For example, in the meta-mapping Σ_B^L of Example 13, since `Capital` $<$ `Gains` $<$ `Sector`, variable A_3 will be bound to `Capital`, excluding it from the target, which involves only A_1 and A_2. Here, the absence of constants in the constraints has two consequences. First, all the laconic mappings that can be generated from a canonical meta-mapping are logically equivalent and so we just need to generate one of them. Second, although we lose the ability to capture the above mentioned domain similarity, this makes the mappings more domain independent, which turns out to be useful in scenarios that have attribute names different from those used in the past.

However, using the above constraints, we can build a large number of different fitting variants out of a single non-fitting meta-mapping. Nevertheless, in Section 6 we show that the presence of variants with different constraints largely increases the chances to find relevant solutions.

In the next section we will present a technique, called *repair*, that turns a non-fitting canonical meta-mapping into a fitting meta-mapping by adding explicit and laconic constraints. Unfortunately, this process unavoidably requires to detect all the ambiguities, a process with inherent exponential complexity (see Appendix B).

Theorem 2. *The problem of finding a fitting meta-mapping for a given canonical meta-mapping and a pair of source and target schemas is* Π_2^p*-hard.*

4 Meta-Mapping Inference

In this section, we present a technique to infer fitting meta-mappings from mappings. Algorithm 1 summarizes our approach: it takes as input a transformation scenario $\mathscr{H}$ and a *strategy*, which specifies whether to infer explicit or laconic mappings, and returns a set of fitting meta-mappings.

Algorithm 1 Meta-mapping inference.

```
1: procedure INFER(ℋ = (M_1, ..., M_n), strategy)
2: (M_i = (S_i, T_i, σ_i). S_i and T_i denote the
3:       m-schema of S_i and T_i, respectively)
4: returns a set of meta-mappings 𝒬 fitting ℋ
5: 𝒬 ← ∅
6:     for M_i in ℋ do
7:       (S_i^{σ_i}, T_i^{σ_i}) ← project-mapping(σ_i, S_i, T_i)
8:       Σ_i ← canonical-meta-mapping(S_i^{σ_i}, T_i^{σ_i})
9:       ℛ_i ← repair(Σ_i, S_i^{σ_i}, T_i^{σ_i}, strategy)
10:        if 𝒬 = ∅ then 𝒬 ← ℛ_i
11:        else 𝒬 ← merge(𝒬, ℛ_i)
12: return 𝒬
```

For each schema mapping σ_i in the scenario, we consider its m-schemas $\mathsf{S_i}$ and $\mathsf{T_i}$ and project σ_i on them (line 7). We then build from $\mathsf{S}_i^{\sigma_i}$ and $\mathsf{T}_i^{\sigma_i}$ the canonical meta-mapping Σ_i (line 8), which may not fit σ_i. We then "repair" it adding constraints (according to the chosen strategy) and produce a set $\mathscr{R}_i$ of meta-mappings, all fitting σ_i (line 9). This step is detailed in Section 4.1.

Since we want to return as output a set $\mathscr{Q}$ of meta-mappings such that every $\Sigma_i \in \mathscr{Q}$ fits $\mathscr{H}$, we also apply a *merge* procedure (line 11) that discards the meta-mappings that do not fit the entire scenario. The details of this step are in Section 4.2.

4.1 Repairing canonical meta-mappings

The repair procedure is in charge of appropriately introducing constraints on the ambiguous variables of the canonical meta-mapping so as to guarantee its fitness.

Explicit Mappings. Algorithm 2 describes how to remove ambiguous variables by adding equalities and inequalities in a meta-mapping $q_S \to q_T$. We compute the set $\mathscr{H}_S$ of all the possible homomorphisms from q_S to source m-schema S^σ and the set $\mathscr{H}_T$ of all the possible homomorphisms from q_T to target m-schema T^σ ($\mathscr{H}_T$) (lines 4-5). Then, we isolate the set $\mathscr{H}_E$ of homomorphisms of $\mathscr{H}_S$ that extend to some homomorphism in $\mathscr{H}_T$ (line 6). Lines 7-8 are dedicated to storing in Γ and Λ the n-uples of potentially ambiguous variables in q_S and q_T respectively, according to Definition 3. The subsequent loop checks whether n-uples of potentially ambiguous variables are indeed ambiguous. The analysis is limited to extending homomorphisms since the others do not produce any

Algorithm 2 Meta-mapping repair.

1: **procedure** EXPLICIT_REPAIR(Σ, S^σ, T^σ)
2: **returns** a set $\mathscr{R}$ of meta-mappings *fitting* σ
3: $\mathscr{R}, \mathscr{A} \leftarrow \emptyset$
4: $\mathscr{H}_S \leftarrow$ generate all the homomorphisms from q_S to S^σ
5: $\mathscr{H}_T \leftarrow$ generate all the homomorphisms from q_T to T^σ
6: $\mathscr{H}_E \leftarrow \{h \in \mathscr{H}_S$ s.t. h *extends* to some $h' \in \mathscr{H}_T\}$
7: $\Gamma \leftarrow \{(x_1, \ldots, x_n)$ of q_S s.t. $\bigcap_{i=1}^n \tau(q_S, x_i) \neq \emptyset\}$, with $x_1 \neq x_2 \neq \ldots \neq x_n$ ▷ set of pot. ambiguos var. in q_S
8: $\Lambda \leftarrow \{(x_1, \ldots, x_n)$ of q_T s.t. $\bigcap_{i=1}^n \tau(q_T, x_i) \neq \emptyset\}$, with $x_1 \neq x_2 \neq \ldots \neq x_n$ ▷ set of pot. ambiguos var. in q_T
9: **for** $h \in \mathscr{H}_E$ **do** ▷ here we detect ambiguous var.
10: **for** $(x_1, \ldots, x_n) \in \Gamma$ **do**
11: **if** $a(h(k_1), \ldots, h(x_1), \ldots, h(k_m)) = \ldots = a(h(w_1), \ldots, h(x_n), \ldots, h(w_m))$
12: for some $(a, \cdot) \in \bigcap_{i=1}^n \tau(q_S, x_i)$ ▷ ambiguity in the LHS
13: **and** $((x_1, \ldots, x_n) \notin \Lambda$
14: **or** $a(h'(k_1), \ldots, h'(x_1), \ldots, h'(k_n)) \neq \ldots \neq a(h'(w_1), \ldots, h'(x_n), \ldots, h'(w_n))$
15: for some $(a, \cdot) \in \bigcap_{i=1}^n \tau(q_T, x_i))$ and h' extension of $h)$ ▷ w/o corresp. ambiguity in the RHS
16: **then** $\mathscr{A} \leftarrow \mathscr{A} \cup \{(x_1, \ldots, x_n)\}$
17: **for** $h \in \mathscr{H}_E$ **do** ▷ here we fix the ambiguities
18: **if** $h(x_1) \neq \ldots \neq h(x_n)$, for all $(x_1, \ldots, x_n) \in \mathscr{A}$ s.t. x_i appears in both q_S and q_T
19: **then**
20: *add* $\forall \mathbf{x}(q_S(\mathbf{x}), \gamma \rightarrow \exists \mathbf{y} q_T(\mathbf{x}, \mathbf{y}))$ to $\mathscr{R}$:
21: where $\gamma = \bigwedge_{(x_1, \ldots, x_n) \in \mathscr{A}} (\gamma_1, \ldots, \gamma_n)$
22: and γ_i is either $x_i = h(x_i)$ or $\bigwedge_{i,j=1\ldots n,\ i \neq j} x_i \neq h(x_j)$.
23: **return** $\mathscr{R}$

tuples in the result. Whenever a homomorphism activates the potential ambiguity in the LHS (lines 11-12), if such ambiguity is not present in the RHS (line 13), or is present but for some extension h', the potentially ambiguous variables participate in atoms that are mapped by h' to different facts (lines 14-15), then the n-uple of ambiguous variables is stored in $\mathscr{A}$. Finally, we exploit the extended homomorphisms in $\mathscr{H}_E$ to fix the ambiguous variables. We isolate the only homomorphisms that have different values for the variables in an ambiguous n-uple (line 18) and, for each of such homomorphisms, we build several repairs by introducing a conjunction of constrains $\gamma_1, \ldots, \gamma_n$ (line 21) for each n-uple of ambiguous variables that appear in both q_S and q_T. Each constraint γ_i can be (line 22): (i) an equality that binds the ambiguous variable x_i to the value $h(x_i)$ assigned by h; (ii) a conjunction of inequalities of the form $x_i \neq h(x_j)$ for any possible ambiguous variable x_j other than x_i. We consider all the possible ways of building each γ_i. Cases in which all the γ_i's have equalities give rise to *positive repairs*; cases with only inequalities give rise to *negative repairs*; otherwise we have *hybrid repairs*.

Example 14. *Assume we want to repair the canonical meta-mapping Σ_B in Example 11.*

Among the homomorphisms in $\mathscr{H}_E$, there are:

$$h_1 : \{(A_1, \mathit{Gains}), (A_2, \mathit{Sector}), (A_3, \mathit{Capital}), \ldots\}$$
$$h_2 : \{(A_1, \mathit{Gains}), (A_2, \mathit{Gains}), (A_3, \mathit{Gains}), \ldots\}$$

The potentially ambiguous variables in q_S and q_T are $\Gamma = \{(A_1, A_2, A_3)\}$ and $\Lambda = \{(A_1, A_2)\}$, respectively. The three Att *atoms of the LHS with the potentially ambiguous variables are all mapped by h_2 to the same fact; moreover, since (A_1, A_2, A_3) is not in Λ, the variables are ambiguous. To repair the meta-mapping, we identify all the extending homomorphisms h_i, s.t. $h_i(A_1) \neq h_i(A_2) \neq h_i(A_3)$. Among them, the homomorphism h_1 above, from which we generate the mappings Σ_B^P, Σ_B^N and Σ_B^H in Example 13.*

In the worst case our technique requires exponential time, as an unavoidable consequence of the hardness of the underlying problem of finding a fitting template mapping (Theorem 2). However, as discussed at the end of the previous section, the large number of discovered mappings is valuable in general, since each of them can be reusable for different scenarios.

Laconic mappings. Differently from explicit mappings, the generation of laconic mappings does not explore all the combinations, thus reducing execution times, and leads to domain independent meta-mappings. Algorithm 3 summarizes our procedure. It scans

Algorithm 3 Generation of laconic meta-mappings.

```
1: procedure LACONIC_REPAIRS(Σ, S^σ, T^σ)
2: returns a set 𝒮 of laconic meta-mappings fitting σ
3:   steps 3-16 of Algorithm 2
4:   for h ∈ ℋ_E
5:     if h(x_1) ≠ ... ≠ h(x_n), for all (x_1, ..., x_n) ∈ 𝒜
6:     then
7:       sort all (x_1, ..., x_n) ∈ 𝒜 by h(x_1), ..., h(x_n)
8:       add the following meta-mapping to 𝒮:
9:       ∀x(q_S(x), γ → ∃y q_T(x, y)),
10:      where γ = ⋀_{(x_1,...,x_n)∈𝒜} (x_1 < ... < x_n),
11:    end if
12: return 𝒮
```

through all the extended homomorphisms of $\mathscr{H}_E$ and stops as soon as we find one that assigns distinct values to the ambiguous variables (line 5). Then, on the basis of the strict total order on the data domains, we sort each n-uple by the value assigned by h to the variables (line 7). The same order is used to produce the constraint on the variables (lines 8-10). We can stop at the first constraint generated since they are all logically equivalent. An example of laconic meta-mapping has been shown in Example 13 (Σ_B^L).

The following result states the correctness of our procedures (proof in Appendix C).

Theorem 3. *Given a canonical meta-mapping Σ defined on m-schemas S^σ and T^σ, a (positive, negative or laconic) meta-mapping Σ_R that fits σ always exists and Algorithms 2 and 3 always terminate and determine such repair.*

4.2 Handling multiple transformations

Let us now come to the problem of generating meta-mappings that fit an entire transformation scenario. To this end, Algorithm 1 *merges* the meta-mappings generated from each schema mapping in the scenario. In order to define the merge operation formally, we first introduce the notion of *meta-mapping extension*:

Definition 5. *Given a meta-mapping* Σ_1 *fitting the mapping* σ_1 *from* $\mathbf{S}_1$ *to* $\mathbf{T}_1$ *and a meta-mapping* Σ_2 *fitting the mapping* σ_2 *from* $\mathbf{S}_2$ *to* $\mathbf{T}_2$*, we say that* Σ_1 extends to Σ_2 (*in symbols* $\Sigma_1 \sqsubseteq \Sigma_2$) *if* Σ_2 *fits* σ_1*.*

To test whether $\Sigma_1 \sqsubseteq \Sigma_2$, we simply apply Definition 1 and check whether $\mathsf{T}_1^{\sigma_1}$ is a universal solution for $\mathsf{S}_1^{\sigma_1}$ under Σ_2. The notion of meta-mapping extension can be used to merge, if possible, sets of meta-mappings.

Definition 6. *The* merge *of two meta-mappings* Σ_1 *and* Σ_2 *is a set of meta-mappings containing: (i)* Σ_2 *if* $\Sigma_1 \sqsubseteq \Sigma_2$*, (ii)* Σ_1 *if* $\Sigma_2 \sqsubseteq \Sigma_1$*, and (iii) both* Σ_1 *and* Σ_2 *if* $\Sigma_1 \sqsubseteq \Sigma_2$ *and* $\Sigma_2 \sqsubseteq \Sigma_1$*. Otherwise the set is empty.*

Note that the notion of meta-mapping extension is implied by logical entailment and therefore *merge* could be as well defined in terms of logical entailment. However, we adopt this more restrictive condition to include in the repository only those mappings that maximize the possibility to find good candidates for new scenarios.

Example 15. *Let us come again the scenario of Figure 1 and consider the fitting meta-mapping* Σ_A *in Example 7. Since* G^{σ_B} *is not a universal solution of* S^{σ_B} *under* Σ_A *(as it can verified by applying the chase on the m-schemas in Example 9) we have that* Σ_A *does not fit* σ_B *and therefore* $\Sigma_B \not\sqsubseteq \Sigma_A$*, which means that the merge would be empty for any extension of* Σ_B *in Example 13. Conversely, it is easy to verify that* $\text{merge}(\Sigma_A,\Sigma_B^N) = \{\Sigma_B^N\}$ *and* $\text{merge}(\Sigma_A,\Sigma_B^H) = \{\Sigma_B^H\}$*, and so both* Σ_B^N *and* Σ_B^H *fit our entire scenario.*

5 The meta-mapping repository

All the meta-mappings inferred from schema mappings are stored in a repository that supports users in the design of data transformations in three use cases: (a) given both source and target schema, generate a ranked list of suitable meta-mappings from the source to the target; (b) given a source schema, generate a ranked list of suitable meta-mappings along with the respective target schemas; and (c) given a target schema, generate a ranked list of suitable meta-mappings along with the respective sources schemas.

A naïve approach to search the repository for suitable meta-mappings could use the same notion of fitness adopted for inference as follows: we scan through all the stored meta-mappings and return those that fit a mapping from a source $\mathbf{S}$ to a target $\mathbf{T}$ by checking whether the m-schema T of $\mathbf{T}$ is a universal solution of the m-schema S of $\mathbf{S}$. However, this approach has some major problems. First, it cannot scale to large repositories since, while the scan is linear, the test above has PTIME complexity as it relies on the chase procedure. Second, it can be applied only if both $\mathbf{S}$ and $\mathbf{T}$ are given, thus only supports setting (a). In addition, the fitness property allows us to retrieve only meta-mappings that closely correspond to the input scenario, neglecting others with slight differences, but maybe relevant. For instance, the meta-mapping in Example 4 is

not suitable for $\mathbf{S}_C$ (Figure 1) because of the different names of the attributes. However, it suffices to change "Profits" into "Gains" in the mapping to successfully reuse it. More sophisticated methods, such as ranking the meta-mappings on the basis of the amount of transferred information [4], are not directly applicable to our settings and have exponential complexity. We therefore propose in this section a heuristic method to overcome the above challenges that allows a ranking of the retrieved meta-mappings and operates in LOGTIME. To this aim, we devise a number of features that capture form and domain similarity of meta-mappings and schemas. Based on these features, we introduce a notion of *coverage* of a meta-mapping and use this notion to rank the meta-mappings.

5.1 Ranking meta-mappings

Coverage is a feature-based metric that enables two kind of comparisons: between a meta-mapping and a database schema and between a pair of meta-mappings. The former ability is used to support the three use cases mentioned above by comparing the LHS and/or the RHS of a meta-mapping with a source and/or a target schema, respectively. The latter is used to organize and index the meta-mappings in the repository by comparing the LHS and RHS of a meta-mapping with the LHS and RHS of another meta-mapping. In the following, we discuss the case of comparing the LHS of a meta-mapping with a source schema, as the other cases are similar.

Coverage is based on the concepts of *structural* and *domain similarity*.

Structural similarity is computed on the basis of a set of scores of features. These features vary from simple, such as the number of atoms in a meta-mapping (or relations in a schema), to more complex, such as the number of unique pairs of atoms in a meta-mapping with at least one common variable (or number of relations in a schema with a homonym attribute). We detail in Appendix D the complete list of these features and their computation. For each feature value f_Σ of the LHS of a meta-mapping, and the corresponding one f_S for the source schema, we compute a $[0, 1]$-normalized distance:

$$d_{f_\Sigma, f_S} = \frac{\mid f_\Sigma - f_S \mid}{\max(f_\Sigma, f_S)}$$

A structural score for each feature is defined as $\sigma_{f_\Sigma, f_S} = 1 - d_{f_\Sigma, f_S}$. Since the features are non-negative, we have $f_\Sigma = f_S = 0$ at most, for which we set $d_{f_\Sigma, f_S} = 0$.

Domain similarity measures how likely the LHS/RHS of a meta-mapping and a schema (or a pair of meta-mappings) deal with the same domain by looking at the presence of shared constants. We say that a constant is shared whenever a variable appearing in an atom of the meta-mapping is bound to a constant and the same constant appears as a "compatible" structural element in the schemas. For example, given the condition $A_1 = \texttt{Gains}$ of a meta-mapping Σ such that A_1 also appears in an atom $\mathsf{Att}(A_1, R)$ of Σ, we have a shared constant with a schema $\mathbf{S}$, if $\mathbf{S}$ has an attribute named `Gains`. The same applies to relation names, keys, and so on. We compute the domain similarity score δ_{MS} as the *Jaccard index* of the sets of constants, i.e., the ratio between the number of shared constants and the overall number of constants.

We interpret each score as the probability that two meta-mappings *cover* each other, in the sense that they are suitable for the same schemas, or that a meta-mapping covers a schema in the sense that it is suitable for it. If a feature has 0.5 score, it is not informative

for the coverage. A lower score indicates that the meta-mapping is less likely to cover the schema, while a higher score indicates that the meta-mapping is more likely to cover the schema. The *coverage* is then built as a combination of all these probabilities. Given σ_{f_Σ,f_S} for structural similarity and $\delta_{\Sigma,S}$ for domain similarity, we combine each of the scores in two steps: *feature scaling* and *Graham combination* [12].

Feature scaling consists in the application to every score of a scaling formula, which adjusts the positive or negative contribution of the score by altering the definition range. Parameters can be set by the user, leveraging domain knowledge. This approach has many degrees of freedom and allows to tune the contribution of the positive and negative influence of the features. We verified experimentally that a good configuration is easy to achieve by hand in a few trials even for complex domains. However, our approach is modular and any parameter estimation technique commonly applied for Bayesian classifiers can be used [7].

At bootstrap, all scores s are in the range $[0, 1]$ and we scale them in the new range $[a, b]$ by replacing s with $a+s\times(b-a)$. Assuming that a score of 0.5 is information neutral, a score > 0.5 testifies a positive influence of the feature comparison on the coverage, and a value < 0.5 testifies a negative influence, we tune a and b for each feature and define a new range as follows: (i) $[0.5, 0.5]$ neutral, (ii) $[0.5, >0.5]$ the score has only a positive effect, (iii) $[<0.5, 0.5]$ the scores has only a negative effect, (iv) $[<0.5, >0.5]$ the score has both positive and negative effects. Finally, we assume the probabilistic independence of the features, perform a Graham combination to consider all the scaled scores, and define the coverage as:

$$\chi = \frac{\prod_{i=1}^{n} \sigma_{f_\Sigma^i,f_S^i} \times \delta_{M,S}}{\prod_{i=1}^{n} \sigma_{f_\Sigma^i,f_S^i} \times \delta_{M,S} + (1-\delta_{M,S}) \times \prod_{i=1}^{n}(1-\sigma_{f_\Sigma^i,f_S^i})}$$

Observe that χ is parametric w.r.t. the scaling parameters $[a, b]$, which tune the influence of the various features.

With the definition of coverage in place, the input schemas are compared against the meta-mappings in the repository and the system returns the top-k by coverage. We introduce next optimization techniques to efficiently retrieve from the repository the most covering meta-mappings.

5.2 Searching meta-mappings

To avoid the exhaustive comparison of the input schema(s) with all the meta-mappings in the repository, we build a *meta-mapping index*. For each meta-mapping added to the repository, we compute the vector of the values of the structural features f_s. Such vector positions the meta-mapping as a point of a multi-dimensional space, which we index by means of a *k-d tree* [5], a space-partitioning data structure that is built in $O(n \log n)$ and enables to retrieve points in a multidimensional space in $O(\log n)$. A k-d tree hinges on a distance function. In our case, we use $1 - \chi_S$, where χ_S is the coverage between two meta-mappings as defined in Section 5.1, where only the structural features are considered.

With the meta-mapping index in place, given an input schema, we: (a) calculate the schema features and build the respective vector; (b) look up such vector in the meta-mapping index to obtain all meta-mappings that are close to the input; (c) calculate the coverage (also considering the domain features) only for these meta-mappings.

We remark the connection between the search operation and the inference mechanism. Our search and ranking techniques can be applied also to a repository of schema mappings rather than to a repository of meta-mappings. While this is simpler to implement (as fitting is trivial), it is less useful in practice. Schema mappings specify a transformation for a specific pair of source and target schemas, while our inferred meta-mappings capture transformations that are independent to some parts of the schemas. We experimentally validated this intuition in the next Section.

6 Experimental evaluation

We evaluate our system, GAIA, on both real-world and synthetic transformation scenarios, as detailed in Section 6.1. In Section 6.2, we report on the efficiency of GAIA in terms of inference and search time for both explicit and laconic meta-mappings. In Sections 6.3 and 6.4, we report on the quality of the returned transformations according to *search accuracy* and *search ability*, respectively. Finally, we compare our approach against a repository of schema mappings in Section 6.5.

6.1 Experimental setup

We implemented GAIA in PL/SQL 11.2 and used Oracle DBMS 11g. All experiments were conducted on Oracle Linux, Intel Core i7@2.60GHz, 16GB RAM.

Use Case (input)	Dataset	Total # st-tgds	Avg # rels	Total # atts	Avg # atoms
S-T	Chamber	20,000	5	900	32
S-T	CDP	1	23	800	15
S-T	Stock	1,300	34	900	40
S	SynthS	21,300	3	800	32
S	SynthT	21,301	12	800	27

Table 1: Transformations and schemas statistics.

Datasets and Transformations. We study three use cases for transformation reuse: given as input schemas **S-T**; only schema **S**, or only schema **T**.

For use case **S-T**, we use a real-world scenario crafted from the data transformations that are periodically done to store data coming from several sources into the *Central National Balance Sheet* (CNBS) database, a national archive of financial information of about 40,000 enterprises. The schema of CNBS is composed of 5 relations with roughly 800 attributes in total. Source data come from three different providers, detailed in the second column of Table 1. The Chamber of Commerce provides data for 20,000 companies (dataset **Chamber**). While the schemas of these datasets are similar to one another, the differences require one mapping per company with an average of 32 (self) joins in the LHS of the transformations involving relations with up to 30 attributes. Data for 20,000 more companies are collected by a commercial data provider (**CDP**) in a single database and then imported into CNBS. The CDP schema is different both in structure and names from the schemas in **Chamber** and requires one mapping involving 15 joins in the LHS. Finally, data for further 1,300 companies is imported from the stock exchange (**Stock**) into CNBS, with each company requiring a mapping with 40 joins on average.

For use case **S**, we use the *synthetic* dataset **SynthS**, which contains 21,300 generated datasets. Each schema has 1-5 relations, randomly extracted from the relations in CNBS; each relation has 2-15 attributes (also randomly extracted), one single key, and one or two

foreign keys. For each synthetic source, we generated the transformation to CNBS (i.e., one target). We use **SynthS** to test the use case where the source is used for searching over the repository in a setting similar to the real-world scenario. The synthetic scenarios are harder than the real-world one, as attributes are randomly selected and combined, thus breaking the semantic associations in the original schemas.

We use **SynthT** to test again use case **S**, but in a scenario with a large number of transformations with different target schemas in the repository. Synthetic dataset **SynthT** contains the 21,300 data sources from **SynthS** and a transformation to a *different*, artificial target schema for each source. We also added a transformation that maps one of the datasets in **Chamber** into CNBS.

For use case **T**, i.e., where only the target schema is given, we rely on **SynthT**. We omit the results for use case **T** when they show the same behaviour of use case **S**; but we report them in Appendix E.

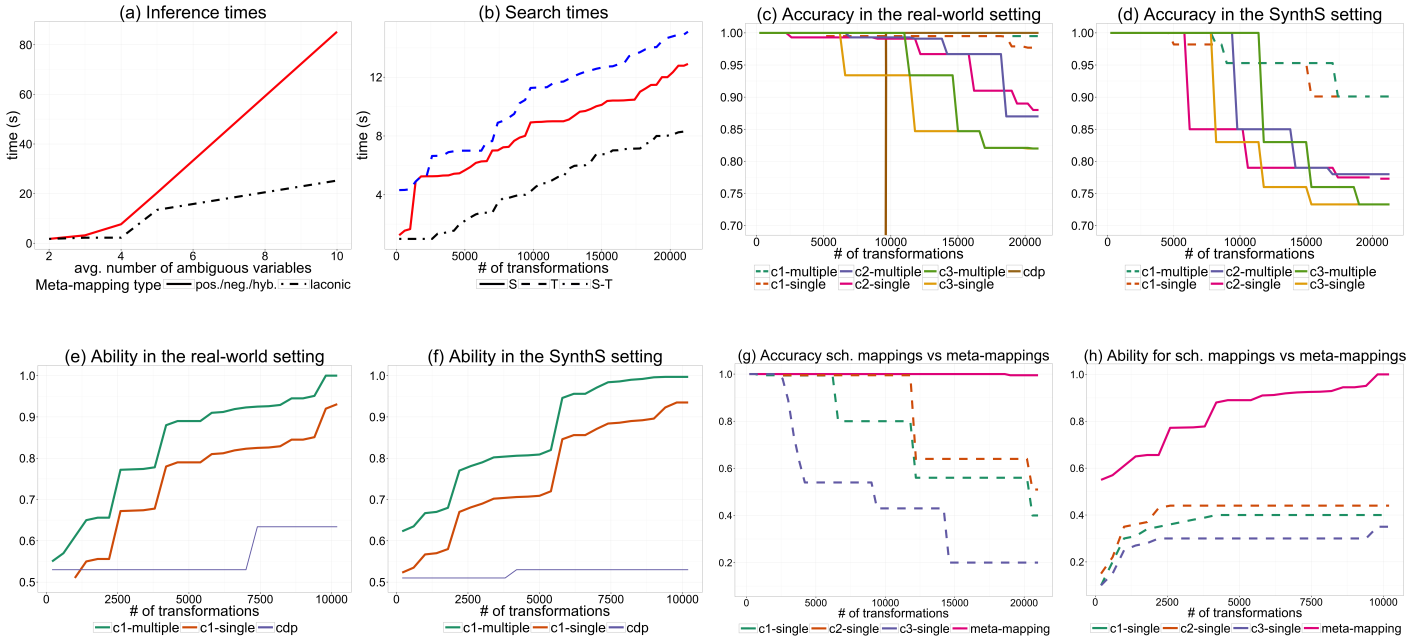

Figure 4: Experimental results on inference times, search times, search accuracy, search ability.

Ranking Strategies. We use three strategies for the ranking of the meta-mappings: strategy **c1** has similar weights for structural and domain similarity, **c2** favors structural similarity, and **c3** privileges domain similarity.

These strategies have been designed to verify the ability of GAIA to leverage different similarity criteria. However, in real applications one configuration suffices for a given repository, depending on its characteristics.

Transformation scenarios. For the inference of the meta-mappings, we consider two configurations: **single**, where a meta-mapping is inferred from one mapping, and **multiple**, where the inference is from a group of mappings. A group contains transformations defined for companies in the same business sector. After testing different numbers of mappings in the group, we observed that the results stabilized with 10 schema mappings

and did not improve significantly with larger numbers, therefore we always consider 10 mappings for the **multiple** configuration.

6.2 Inference and search times

In Figure 4(a), we report the times for computing the inference of meta-mappings. We run the test inferring the meta-mappings from mappings taken from SynthT with an increasing number of ambiguous variables (x-axis: 2-10), hence requiring the evaluation of an exponentially growing number of homomorphisms. When given 20,000 mappings and 10 ambiguous variables, about 21 millions explicit meta-mappings (20,000 laconic) are stored in the repository. For each mapping, we generate both explicit and laconic meta-mappings. For a large number of variables, explicit meta-mappings become time consuming, taking more than a minute for 10 ambiguous variables, while laconic meta-mappings take about 20 seconds.

We then evaluate the search time for the three use cases on the same repository, as reported in Figure 4(b). For each test, we average the response times of 25 randomly selected tests. From the results, we observe low latency (in the order of seconds) even for large repositories generated from more than 20,000 input mappings with millions of meta-mappings in the repository. Search times confirm the logarithmic behaviour of the search using the index over meta-mappings. Interestingly, in use case **S** we have better performances than in use case **T**. This difference is due to a less selective index for **T**, which returns bigger clusters and requires more comparisons to rank the output. The artificial generation of the target schemas from a single schema (CNBS) produces a more uniform distribution, which, in turn, makes the meta-mappings structurally similar. As expected, more information in the index lookup leads to a more selective search, thus use case **S-T** has the best performance.

6.3 Search accuracy

For each mapping σ, defined by the user on a source S and a target T, the repository stores a set of new meta-mappings $\mathscr{Q}$, inferred from σ. We measure the search *accuracy* in terms of the ability of the system to return the meta-mappings in $\mathscr{Q}$ when queried with S, T, or both, i.e., it measures how well the system retrieves known cases.

We use a *resubstitution* approach: we populate an empty repository by inferring the meta-mappings from a set Θ of schema mappings. For each schema mapping σ, we store the generated meta-mappings $\mathscr{Q}_\sigma$. We then pick a schema mapping σ' from Θ and search the repository by providing as input either the source, target, or both schemas in σ' as input. As a query returns thousands of applicable meta-mapping, we rank them according to the coverage and take the top 10. We collect the top-10 transformations and compute the percentage of rightly retrieved meta-mappings Σ, i.e., $\Sigma \in \mathscr{Q}_{\sigma'}$. We randomly pick 50 schema mappings from Θ and report the average accuracy over them.

We test search accuracy with a repository of increasing size (from 200 to 21,301), both on real (Figure 4(c)) and synthetic mappings (Figure 4(d)). In each test, we query the repository by using 50 different source schemas in SynthS for use case **S**, 50 different source-target pairs in the real scenario for use case **S-T**, and report their average accuracy with ranking strategies **c1**, **c2** and **c3**. The experiment is repeated for single and multiple scenarios. The largest corpus contains about 700K explicit meta-mappings for single

configuration (110K for multiple configuration) and about 21K laconic meta-mappings for single configuration (5K for multiple). In the multiple scenario, the test is considered successful on a mapping σ when the retrieved meta-mapping originates from the group that includes σ.

As we see in Figure 4(c-d), accuracy decreases with the size of the repository as larger numbers of stored transformations lead to an increasing number of false positives in the search result. Strategy **c1** shows the best accuracy results since it exploits both structural and domain properties: 0.97 in the real setting and 0.9 in the synthetic setting for repositories with 21K mappings. Strategies **c2** and **c3** perform worse, but accuracy never goes below 0.82 in the real setting and 0.733 in the synthetic one.

We report in Figure 4(c) the ability to retrieve the correct mapping for CDP with a separate line. It is interesting to see that as soon as CDP mapping gets inserted in the repository (after about 10,200 transformations), it is immediately identified by the search with all configurations. This shows that very specific structures and constants lead to right meta-mappings that are easy to retrieve. While the general trends for real and SynthS scenarios are similar, in SynthS the accuracy tends to decrease faster. This is because less information is available when the target is not provided and synthetic schemas are generated by sampling the original datasets, thus original semantic associations in the schemas are broken. If associations between attributes are lost, it is harder to identify the correct mapping and false positives increase. For accuracy test in SynthT setting, we query providing the only real source (from Chamber) in the setting with an increasing number of meta-mappings in the repository. Similarly to the CDP case, the correct mapping is immediately identified as soon as the meta-mapping for CNBS is added to the repository.

Finally, we remark that inferring meta-mappings from multiple schema mappings (multiple scenario) improves accuracy for two main reasons. The meta-mappings are more general since generated from a larger number of similar cases, reducing the risk of overfitting. Also, the repository size decreases for all configurations.

6.4 Search ability

We now evaluate GAIA in the search of valuable transformations for new schemas, i.e., scenarios (and therefore meta-mappings) that have not been seen by the system yet. For this task, we introduce the notion of *search ability*. Given a mapping σ from S to T and the set of meta-mappings $\mathscr{Q}_\sigma$ inferred by σ, we consider S, T, or both and search the repository, making sure that it does not include any meta-mapping of $\mathscr{Q}_\sigma$. We then measure the search ability as the coverage (as defined in Section 5.1) of the best retrieved meta-mappings w.r.t. the meta-mappings in $\mathscr{Q}_\sigma$.

In the experiment, we resort to a *hold-one-out* technique. We start from the empty repository and a set Θ of mappings. We populate the repository with a set $\Gamma \subset \Theta$ of mappings and then randomly choose a set of mappings $\Phi \subset \Theta - \Gamma$ (the "new mappings"). For each $\sigma \in \Phi$ from S to T, we query the repository giving S, T, or both as input. We then compute the average α of the coverage of the top-10 meta-mappings w.r.t. the meta-mappings $\mathscr{Q}_\sigma$. Finally, we compute the average of the α values of all $\sigma \in \Phi$.

We test the dependence of search ability on the size of the repository and the transformation scenario. We report only on ranking strategy **c1**, which, as we have seen for the accuracy, is the most comprehensive. For each test run, the repository is populated

with the meta-mappings generated by a number of transformations (from 200 to 10,200), in which we do not include CDP, in order to assess how well the system can provide good transformations for it. We then compute the search ability with 50 different source schemas and 50 different source-target pairs. In this case, we explicitly run the experiment using CDP. All tests are repeated in the real, SynthS, and SynthT settings.

As apparent in Figure 4(e-f), the more mappings are added to the repository, the higher is the search ability. This is expected, as the presence of more meta-mappings increases the likelihood to find a suitable one for the given input. For CDP, we report only the best result, which is obtained with the multiple configuration. We observe a significant improvement in the real-world setting after the inclusion of transformation $7,400$, which is closer to the size of CDP.

Grouping mappings as a single transformation scenario (multiple configurations) improves performance also in this case. Not only the size of the repository is reduced, but for the same number of input mappings, we observe an increased search ability for all configurations in both settings. This shows the positive impact of the merge procedure.

The slightly worse search ability in SynthS (f) is again due to the missing information about the desired target and the broken semantic associations. For SynthT (and use case **T**), we verified a similar ability to provide good mappings given a source (target) schema similar to the real one.

Our tests on search ability show that very complex mappings can be hardly approximated by meta-mappings that do not originate from them. Conversely, simpler transformations can be approximated very well with a sufficiently large set of meta-mappings.

6.5 Schema Mappings Corpus

Finally, we repeat the experiments on the real-world dataset and use case **S-T** to measure search accuracy and ability of our system when the repository is populated with schema mappings. To build a schema mapping repository we compute the canonical meta-mapping for each schema mapping and bind all the variables to the specific constants, independently of their ambiguity, to guarantee fitness. The results for the meta-mappings are the ones we obtained in the previous experiments with **c1** and multiple configurations.

In Figure 4(g), the accuracy results with a schema mapping repository and strategy **c2**, which exploits the structure of dependencies, are comparable to meta-mappings up to about 13,000 input mappings. Then the responses become inaccurate because on the scarce generality of schema mappings, which lead to many false positives in the search. The differences between the two approaches become more apparent with the search ability (Figure 4(h)). For new, unseen cases, a repository of schema mappings does not provide significant benefit, never going above 0.4 for any configuration. This is also due to the specificity of schema mappings, that are hardly applicable for semantically similar cases.

7 Related Work

The present work makes use of a framework for dealing with generic schema mappings that has been proposed in [18]. However, the only material taken from the earlier work is confined to Section 2. The rest of the paper present new ideas and methods. The approach proposed here can be adopted using other formalisms for meta-mappings [14].

While the notion of reuse is popular for software components, it is getting growing attention in the database community only recently [1]. The motivation comes from the increased capability of storing metadata and "enterprise knowledge" in non-structured repositories, often referenced as *data lakes* [10, 13] and from the wish to use such knowledge in data preparation tasks (data wrangling) [11]. Our work goes in this direction by introducing a general framework for the reuse of schema mappings expressing data transformations.

In our approach, as in traditional model management [6], schemas, mappings, and meta-mappings are treated as first class citizens. The language adopted for the declarative definition of data transformations is the standard for data exchange [8, 16], but meta-mappings exploit a semantics for the problem of exchanging metadata [18], instead of data. One important contribution in data exchange is the semi-automatic generation of the schema mappings [9], given simple correspondences between the schema elements (possibly discovered automatically with a schema matching tool [19]). The combination of matching discovery and mapping generation is of great support for data transformation design [15], but it is applicable only when source and target schemas are given, while we handle also the case with source or target only. Moreover, schema mappings are usually manually tuned with extra information coming from the users' background knowledge. This refinement is a valuable human effort that our system is able to reuse. There exists proposals to store element matches for reuse [17], but they cannot store logical formulas and therefore crucial information, such as the structure of the schema and the manual tuning, is completely lost. Some proposals [3] focus on a framework to support the user in the design of mappings by means of data examples. This is valuable, yet does not deal with the reuse of early designed mappings or knowledge information.

Traditional ETL tools allow to store and load previously define transformations, without any schema-level generalization or understanding of the transformation semantics.

As shown in the experiments, our inference algorithm increases the chances of reusing previous transformations. The inference of meta-mappings from mappings can be seen as the lifting of the discovery of mappings from data example [2, 3]. However, in our setting, these algorithm would lead to canonical meta-mappings that, as discussed in Section 3, are not fitting and therefore do not precisely capture the semantics of a transformation. Our repair algorithms (Section 4.1) can be seen as a generalization of the previous algorithm [3] where, with a limited extension of the adopted language, we also deal with data examples with self-joins.

8 Conclusions

We introduced a system to support the design of data transformations. Starting from a set of schema mappings, we generate more generic meta-mappings, which capture the semantics of the input transformations at a higher level of abstraction. We store meta-mappings in a searchable repository and index them for fast retrieval. Given a new scenario, we provide a list of "suitable" schema mappings, for an input source schema, an input target schema, or for a pair of source and target schemas. Our experiments, on both real and synthetic datasets, confirm the high quality of the retrieved mappings.

One possible direction for future work is the extension of the approach to take into account constants and function symbols in the schema mappings. This would allow us to

capture and reuse more information, such as data-level constraints that are meaningful to the user.

A Proof of Theorem 1

Theorem 1. *A canonical meta-mapping Σ fits its mapping σ if and only if it does not contain ambiguous variables.*

Proof. Let S^σ, T^σ be the m-schemas on which Σ is defined, which we called data examples.

($\Rightarrow$) Let us first show that if Σ does not have ambiguous variables, then it fits its data examples. We have to prove that, in this case, T^σ is a universal solution for S^σ under Σ. To show that T^σ is a a solution for S^σunder Σ, we show that every homomorphism h from q_S to S^σ extends to some homomorphism h' from $\mathsf{S}^\sigma \cup \mathsf{T}^\sigma$ to $q_S \wedge q_T$. Let us proceed by contradiction. The only case in which this extension is impossibile takes place when an atom a of q_S with variable x_i is mapped by h into a fact $a(\ldots, h(x_i) = X_i, \ldots)$ of S^σ, variable x_i also appears in q_T in atom b and all homomorphisms h' from q_T to T^σ map b into facts $b(\ldots, h'(x_i) = X_j, \ldots)$, with $X_i \neq X_j$. Thus, by construction of canonical meta-mappings, since atoms a and b share a variable, there exists another fact $a(\ldots, X_j, \ldots)$ of S^σ and another atom in q_S, such that for some homomorphism h'', we have $a(\ldots, h''(x_j) = X_j, \ldots)$, with $x_i \neq x_j$. Therefore, it holds $\tau(q_S, x_i) \cap \tau(q_S, x_j) \neq \emptyset$ and h'' can extend to h'. The two previous conditions show that x_i and x_j are actually ambiguous according to Definition 4 and contradict the hypothesis, therefore proving that T^σ is a solution for S^σ under Σ. Moreover, since there is a fact in T^σ for each atom in q_T, by definition of canonical meta-mapping, T^σ is the most general, i.e. universal, solution for T^σ under Σ.

($\Leftarrow$) Let us now show that if Σ fits its data examples, then it does not have any ambiguous variables. Let us proceed again by contradiction. Let x_i and x_j, with $x_i \neq x_j$, be ambiguous variables in q_S. Thus, by Definition 4, there are two homonym and homomorphic ($\tau(q_S, x_i) \cap \tau(q_S, x_j) \neq \emptyset$) atoms a in q_S, one with x_i and one with x_j, such that for some homomorphism h from q_S to S^σ, it holds that $a(\ldots, h(x_i) = X_i, \ldots)$ and $a(\ldots, h(x_j) = X_i, \ldots)$. Moreover, one of the following two conditions holds:

- either x_i or x_j (or both) appear in q_T and they are not potentially ambiguous. Thus there is an atom $b(\ldots, x_i, \ldots)$ in q_T such that $b(\ldots, h(x_i) = X_i, \ldots)$, without any other homomorphic and homonym atoms in q_S having x_j. Now, by definition of canonical meta-mapping, there is another fact $a(\ldots, X_j, \ldots)$ in S^σ, with $X_i \neq X_j$ and for some homomorphism h'', we have $a(\ldots, h''(x_j) = X_j, \ldots)$. Since b has no homomorphic and homonym atoms having x_j, we have that h'' does not extend to any h' from $\mathsf{S}^\sigma \cup \mathsf{T}^\sigma$ to $q_S \wedge q_T$. Therefore T^σ is not a universal solution for S^σ under Σ, and Σ does not fit its data examples. This contradicts the hypothesis, which implies that Σ does not have ambiguous variables.

- x_i and x_j appear in q_T in two homonym and homomorphic atoms $b(\ldots, x_i, \ldots)$ and $b(\ldots, x_j, \ldots)$ and there is an extension h' of h such that the two are mapped into different facts (for which the assignment $h'(x_i) = h'(x_j) = X_i$ holds by definition of extension). Now, by definition of canonical meta-mapping, there is another fact $a(\ldots, X_j, \ldots)$ in S^σ, with $X_i \neq X_j$ and for some homomorphism h'', we have

$a(\ldots, h''(x_j) = X_j, \ldots)$. As we have seen for h', atoms homomorphic and homonym to $b(\ldots, x_j, \ldots)$ map into two possible different facts of the form $b(\ldots, X_i, \ldots)$. Moreover, the two atoms $b(\ldots, x_j, \ldots)$ can map only to those two facts[2] by definition of canonical meta-mapping. Therefore h'' does not extend to any h' from $\mathsf{S}^\sigma \cup \mathsf{T}^\sigma$ to $q_S \wedge q_T$, contradicting the hypothesis and proving that Σ does not have ambiguous variables.

□

B Proof of Theorem 2

Theorem 2. *The problem of finding a fitting meta-mapping for a given canonical meta-mapping and a pair of source and target m-schemas is* Π_2^p*-hard.*[3]

Sketch. The generation of canonical meta-mappings is linear in the size of m-schemas since one needs to add to the mapping one atom for each fact of the source (and target) m-schemas. On the contrary, since the number of possible homomorphisms between q_s and S^σ (and between q_T and T^σ) can be exponential, any procedure that needs to discover the ambiguous variables will run in exponential time in the worst case. Moreover, this worst case scenario is always realized, as in order to detect the ambiguities, one cannot decide to stop analyzing homomorphisms, but must discover all the ones in $\mathscr{H}_S$ that extend to some in $\mathscr{H}_T$. This involves a universal quantification of homomorphisms, followed by an existential quantification and therefore places the problem in the Π_2^p class. □

C Proof of Theorem 3

Theorem 3. *Given a canonical meta-mapping* Σ *defined on m-schemas* S^σ *and* T^σ*, a (positive, negative or laconic) meta-mapping* Σ_R *that fits* σ *always exists. Moreover, our procedures (Algorithms 2 and 3) always terminate and determine such repair.*

Proof. In absence of ambiguous variables, Σ already fits σ by Theorem 1. Let us turn to the case in which ambiguous variables are present. By definition of canonical meta-mapping, Σ has one atom for each fact in S^σ and T^σ. Therefore, there exists a homomorphism h from q_S to S^σ that extends to some h' from $q_S \wedge q_T$ to $\mathsf{S}^\sigma \cup \mathsf{T}^\sigma$ and for which $h(x_1) \neq \ldots \neq h(x_n)$ holds for all the n-uples of ambiguous variables. A fitting-meta mapping Σ_R can be thus always determined by constraining each of the ambiguous variables x_i to $h(x_i)$. Since Σ_R allows all the homomorphisms allowed by Σ, except for the ones producing undesired facts, it follows that T^σ is the universal solution of S^σ under Σ_R, that is, Σ_R fits σ. If there are no ambiguous variables, our procedures (Algorithms 2 and 3) immediately terminate; conversely, each homomorphism h is determined by exhaustively evaluating $h(x_1) \neq \ldots \neq h(x_n)$ on extending homomorphisms. Since they are finite, the procedure terminates. Moreover, they return correct repairs for Σ as in facts they constrain each ambiguous variable x_i exactly to the value $h(x_i)$ (with the three different type of constraints we have illustrated). □

[2]Indeed, here for the sake of simplicity we consider only pairs of ambiguous variables, but the proof can be easily generalized to n-uples.

[3]Π_2^p-completeness could be also proved by reduction from the evaluation problem for quantified Boolean formulas.

D Features

Feature	Description
L_REL	number of atoms in the LHS (relations in the source schema).
R_REL	number of atoms in the RHS (relations in the target schema).
EXIST	number of existentially quantified variables (constructs in the target schema whose name never appears in the source).
L_JOIN	number of unique pairs of atoms in the LHS with at least one variable in common (relations in the source schema with at least one attribute in common).
R_JOIN	number of unique pairs of atoms in the RHS with at least one variable in common (relations in the target schema with at least one attribute in common).
L_CART	number of unique pairs of atoms in the LHS without any variables in common (relations in the source schema with no attributes in common).
R_CART	number of unique pairs of atoms in the RHS without any variables in common (relations in the target schema with no attributes in common).
L_JOIN_FK	number of unique pairs of atoms in the LHS with at least one variable in common and this variable appears in a FKEY atom (relations in the source schema with common foreign-key related attributes).
L_CART_FK	number of unique pairs of atoms in the LHS, linked by a FKEY, without any variables in common (relations in the source schema without common attributes and nevertheless linked by a foreign key).
R_JOIN_FK	number of unique pairs of atoms in the RHS with at least one variable in common and this variable appears in a FKEY atom (relations in the source schema with common foreign-key related attributes).
R_CART_FK	number of unique pairs of atoms in the RHS, linked by a FKEY, without any variables in common (relations in the target schema without common attributes and nevertheless linked by a foreign key).
L_JOIN_KEY	number of unique pairs of atoms in the LHS with at least one variable in common and this variable appears in a KEY atom (relations in the source schema with common key attributes).
R_JOIN_KEY	number of unique pairs of atoms in the RHS with at least one variable in common and this variable appears in a KEY atom (relations in the target schema with common key attributes).
VAR_COPIED	number of distinct variables copied from the LHS to the RHS (number of homonym constructs in the source and target schema).
VAR_JOINED	number of unique pairs of variables in distinct atoms, not involved in any FKEY in the LHS, copied to the same atom or into a pair of atoms linked by a FKEY in the RHS (number of unique pairs of attributes in distinct relations in the source, not involved in any foreign key, that appear in a pair of relations linked by a foreign key in the target schema).
VAR_DISJOINT	number of unique pairs of variables in the same atom in the LHS, or in atoms linked by a FKEY, copied to distinct atoms, not linked by a FKEY in the RHS (number of unique pairs of attributes in the same relation in the source that appear in a pair of distinct relations not linked by a foreign key in the target schema).
VAR_NORMALIZED	number of unique pairs of variables in the same atom in the LHS, copied to distinct atoms in the RHS, linked by a FKEY (number of unique pairs fo attributes in the same relation in the source that appear in a pair of distinct relations linked by a foreign key in the target schema).
VAR_DENORMALIZED	number of unique pair of variables in distinct atoms in the LHS, linked by a FKEY, copied to the same atom in the RHS (number of unique pairs fo attributes in distinct relations in the source, linked by a foreign key, that appear in the same relation in the target schema).

Figure 5: The structural features that we use to evaluate the coverage.

Figure 5 shows the complete list of the structural features that we use to calculate

the coverage of a meta-mapping with respect to a schema or between meta-mappings. Some features specifically refer to the LHS or the RHS of the meta-mapping, which must be interpreted as the source and target schemas, respectively, when applied to schema. Clearly, settings where only the source (target) schemas are present use only a subset of the features.

E Results for the SynthT scenario

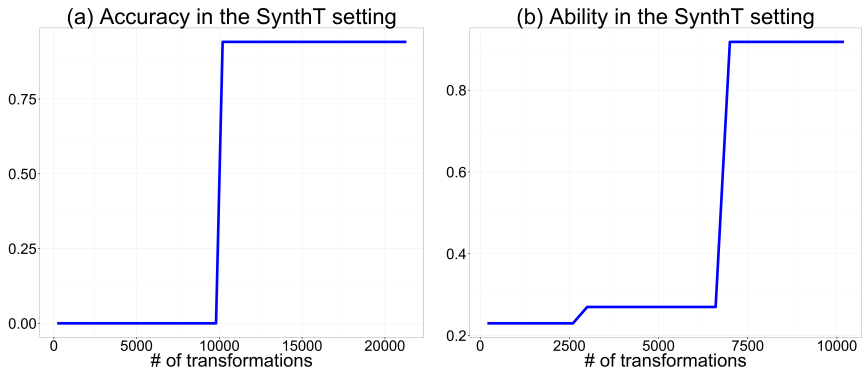

Figure 6: Experimental results on search accuracy (a) and search ability (b) for the setting **SynthT**.

In Figure 6 we report the experimental results on search accuracy and ability for the setting **SynthT**.

F Configurations

Figure 7 reports the configurations used in the experiments. They correspond to three different strategies in ranking the meta-mappings. Configuration **c1** assigns similar weights to both structural and domain similarity so that both are taken into high consideration. Configuration **c2** privileges structural features, which will prevail in computing the coverage, even having negative influence. Finally, configuration **c3** privileges structural similarity, which has both negative and positive influence; on the other hand, structure has almost no effects on coverage as the parameters are neutral. Observe that the different strategies (privilege the structure, the domain or both) have been followed to highlight the behavior of GAIA in terms of the quality of the results when the parameters vary. The values in Figure 7 have been determined in a trial-and-errors iterative process and, as we have shown in the experiments, some configurations perform better than others. For real-life operation, a single GAIA repository requires only one specific tuning, depending on its characteristics.

Parameters	Configurations					
	c1		c2		c3	
	a	b	a	b	a	b
L_REL	0.4	0.75	0.35	0.75	0.5	0.55
R_REL	0.4	0.60	0.35	0.75	0.5	0.55
EXIST	0.4	0.60	0.35	0.65	0.5	0.55
L_JOIN	0.5	0.57	0.38	0.58	0.5	0.65
R_JOIN	0.5	0.55	0.38	0.55	0.5	0.65
L_CART	0.5	0.55	0.4	0.57	0.5	0.55
R_CART	0.5	0.55	0.5	0.55	0.5	0.65
L_JOIN_FK	0.5	0.55	0.5	0.55	0.5	0.5
L_CART_FK	0.5	0.55	0.5	0.55	0.5	0.5
R_JOIN_FK	0.5	0.55	0.5	0.55	0.5	0.5
R_CART_FK	0.5	0.55	0.5	0.55	0.5	0.5
L_JOIN_KEY	0.5	0.58	0.5	0.58	0.5	0.5
R_JOIN_KEY	0.5	0.58	0.5	0.58	0.5	0.5
VAR_COPIED	0.5	0.65	0.5	0.65	0.5	0.58
VAR_JOINED	0.5	0.55	0.5	0.55	0.5	0.58
VAR_DISJOINT	0.5	0.53	0.5	0.53	0.5	0.54
VAR_NORMALIZED	0.5	0.56	0.5	0.56	0.5	0.5
VAR_DENORMALIZED	0.5	0.53	0.5	0.53	0.5	0.5
Domain affinity	0.4	0.85	0.5	0.63	0.3	0.85

Figure 7: The three configurations we used in the experimental settings.

References

[1] Daniel Abadi et al. The Beckman report on database research. *Commun. ACM*, 59(2):92–99, January 2016.

[2] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23, 2011.

[3] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Designing and refining schema mappings via data examples. In *SIGMOD*, pages 133–144, 2011.

[4] Marcelo Arenas, Jorge Perez, Juan L. Reutter, and Cristian Riveros. Foundations of schema mapping management. In *PODS*, pages 227–238, 2010.

[5] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[6] Philip A. Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, 2007.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, 2003.

[9] Ronald Fagin, Laura M. Haas, Mauricio A. Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling*, 2009.

[10] Raul Castro Fernandez, Ziawasch Abedjan, Samuel Madden, and Michael Stonebraker. Towards large-scale data discovery. In *ExploreDB*, pages 3–5, 2016.

[11] Tim Furche, Georg Gottlob, Leonid Libkin, Giorgio Orsi, and Norman W. Paton. Data wrangling for big data: Challenges and opportunities. In *EDBT*, pages 473–478, 2016.

[12] Paul Graham. Better bayesian filtering. In *Proceedings of Spam Conference*, 2003.

[13] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Managing Google's data lake: an overview of the goods system. *IEEE Data Eng. Bull.*, 39(3):5–14, 2016.

[14] Mauricio A. Hernández, Paolo Papotti, and Wang Chiew Tan. Data exchange with data-metadata translations. *PVLDB*, 1(1):260–273, 2008.

[15] Verena Kantere, Dimos Bousounis, and Timos K. Sellis. A tool for mapping discovery over revealing schemas. In *EDBT*, 2009.

[16] Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *ACM PODS*. ACM, 2005.

[17] Jayant Madhavan, Philip A Bernstein, AnHai Doan, and Alon Halevy. Corpus-based schema matching. In *ICDE*. IEEE, 2005.

[18] Paolo Papotti and Riccardo Torlone. Schema exchange: Generic mappings for transforming data and metadata. *Data Knowl. Eng.*, 68(7):665–682, 2009.

[19] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[20] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. Express: A data extraction, processing, and restructuring system. *TODS*, 2(2):134–174, 1977.