Meta-Mappings for Schema Mapping Reuse

Paolo Atzeni
Università Roma Tre
paolo.atzeni@uniroma3.it

Luigi Bellomarini
Banca d'Italia
luigi.bellomarini@bancaditalia.it

Paolo Papotti
EURECOM
papotti@eurecom.fr

Riccardo Torlone
Università Roma Tre
riccardo.torlone@uniroma3.it

ABSTRACT

The definition of mappings between heterogeneous schemas is a critical activity of any database application. Existing tools provide high level interfaces for the discovery of correspondences between elements of schemas, but schema mappings need to be manually specified every time from scratch, even if the scenario at hand is similar to one that has already been addressed. The problem is that schema mappings are precisely defined over a pair of schemas and cannot directly be reused on different scenarios. We tackle this challenge by generalizing schema mappings as *meta-mappings*: formalisms that describe transformations between generic data structures called *meta-schemas*. We formally characterize schema mapping reuse and explain how meta-mappings are able to: (i) capture enterprise knowledge from previously defined schema mappings and (ii) use this knowledge to suggest new mappings. We develop techniques to infer meta-mappings from existing mappings, to organize them into a searchable repository, and to leverage the repository to propose to users mappings suitable for their needs. We study effectiveness and efficiency in an extensive evaluation over real-world scenarios and show that our system can infer, store, and search millions of meta-mappings in seconds.

PVLDB Reference Format:
P. Atzeni, L. Bellomarini, P. Papotti, R. Torlone. Meta-Mappings for Schema Mapping Reuse. *PVLDB*, 12(5): 557-569, 2019.
DOI: https://doi.org/10.14778/3303753.3303761

1. INTRODUCTION

Schema mappings are widely used as a principled and practical tool for data exchange and data integration. However, although there are systems supporting data architects in the creation of mappings [12, 14], designing them is still a hard and time-consuming task. In this framework, it has been observed that, given the overwhelming amount of "enterprise knowledge" stored in traditional data warehouses and in data lakes [16, 21, 31], reuse is an opportunity of increasing importance [1]. In particular, data transformation scenarios are often defined over schemas that are different in structure but similar in semantics. This is especially true if data sources, which are extremely heterogeneous, have to be mapped to a shared format. In this setting, it often happens that mappings are shared as well, so that previous experience can be leveraged. It follows that a great opportunity to reduce the effort of transformation design is to *reuse existing schema mappings*. Unfortunately, there is no obvious approach for this problem. Consider the following example inspired from a real use case.

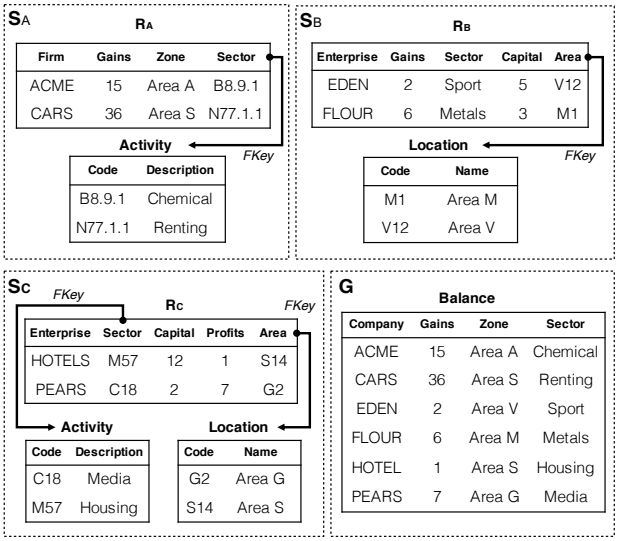

Figure 1: A data transformation scenario

EXAMPLE 1. *A central bank maintains a register with balance data from all companies in the country (Figure 1). This register has schema* $\mathbf{G}$*, with a 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* $\mathtt{R_A}$ *for companies (firms), with gains, zone of operation, and economic sector, whose code refers to relation* `Activity`*. Provider B adopts a schema* $\mathbf{S}_B$*, with a relation* $\mathtt{R_B}$ *for companies (enterprises), their gains, sector, capital, and area, whose code refers to relation* `Location`*. Data is moved from* $\mathbf{S}_A$ *and* $\mathbf{S}_B$ *into* $\mathbf{G}$*, by using two schema mappings:*

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

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

Proceedings of the VLDB Endowment, Vol. 12, No. 5
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3303753.3303761

The example shows a *data exchange scenario* where the differences in the mappings are due to the structural differences between $\mathbf{S}_A$ and $\mathbf{S}_B$, which are, on the other hand, semantically very similar. Moreover, every new data provider (e.g., $\mathbf{S}_C$ in the Figure) would require the manual design of a new, ad-hoc mapping, even if there is a clear analogy with the already defined mappings.

So, what is the right way to reuse σ_A and σ_B and avoid the definition of a new mapping for $\mathbf{S}_C$? Ideally, we would like to collect all the available mappings in a repository; then, for any new pair of schemas (e.g., $\mathbf{S}_C$ and $\mathbf{G}$ in the Figure), we would like to query such repository to retrieve a suitable mapping. Unfortunately, this form of direct reuse is complicated by the nature of schema mappings. Actually, a mapping characterizes integration scenarios as a constraint and captures the semantics for a pair of schemas at a level of detail that enables both logical reasoning and efficient execution. Yet, a simple variation in a schema, such as a different relation or attribute name or a different number of attributes, makes it not applicable. Indeed, our experiments show that mappings of a corpus of 1.000 schema mappings can be reused for new, unmapped pairs of schemas only in 20% of cases. To be reusable, a mapping should be described in a way that is independent of its specificities but, at the same time, harnesses the essence of the constraint so as to work for similar schemas.

EXAMPLE 2. *Consider a "generic" mapping Σ_A, obtained from σ_A by replacing names of the relations and attributes with variables. It 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 attribute* d
    with a foreign key constraint from s *of* r *to* s *of* r'
  there exists *a relation* r'' with *key* f *and attributes* g, a, d.

If instantiated on $\mathbf{S}_A$, the generic mapping Σ_A expresses a mapping to $\mathbf{G}$ that is the same as σ_A. This solution seems a valid compromise between precision, i.e., the ability to express the semantics of the original mapping, and generality, as it can be applicable over different schemas. However, Σ_A falls short of the latter requirement, as it is not applicable on $\mathbf{S}_B$. Indeed, there are no constraints on attribute g and a, and so they could be bound to any of `Gains`, `Sector` and `Capital`, incorrectly trying to map `Capital` into the target.

EXAMPLE 3. *Consider now a more elaborated generic mapping that uses constants to identify attributes:*

Σ_B^H*:* for each *relation* r with *key* e *and attributes* g, s, c, a
  for each *relation* r' with *key* a *and attribute* d
    with a foreign key constraint from a *of* r *to* a *of* r'
    where $g = Gains$, $s \neq Gains$, $s \neq Capital$,
      $c \neq Gains$, $c \neq Sector$
  there exists *a relation* r'' with *key* e *and attributes* g, d, s.

This generic mapping is precise enough to correctly describe both σ_A and σ_B and can be re-used with other schemas.

The example shows a combination of attributes, identified by constraints on their names and role, that form a correct and useful generic mapping. Once pinpointed, generic mappings can be stored in a repository, so that it is possible to retrieve and use them for a new scenario. In our example, given $\mathbf{S}_C$ and $\mathbf{G}$, the generic mapping Σ_B^H can be retrieved from the repository and immediately applied.

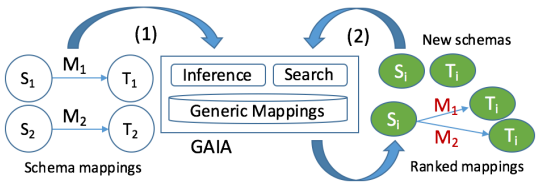

Figure 2: The architecture of GAIA.

There are three main challenges in this approach.

- We need a clear characterization of what it means for a generic mapping to correctly describe and capture the semantics of an original schema mapping.
- A generic mapping is characterized by a combination of constraints (e.g., the ones on attribute names and roles) and so, for a given original mapping, there is a combinatorial blow-up of possible generic mappings. We need a mechanism to generate and store all of them.
- For a new scenario (e.g., new schemas), there is an overwhelming number of generic mappings that potentially apply, with different levels of "suitability". We need efficient tools to search through and choose among them.

In this work, we address the above challenges with GAIA, a system for mapping reuse. GAIA supports two tasks, as shown in Figure 2: (1) infer generic mappings, called *meta-mappings*, from input schema mappings, and store them in a repository; (2) given a source and a target schema, return a ranked list of meta-mappings from the repository which are used to generate possible mappings between these schemas.

In sum, this paper provides the following contributions:

- The notion of *fitness*: a semantics to precisely characterize and check when a meta-mapping is suitable for a reuse scenario (Section 3).
- A necessary and sufficient condition for the fitness of a meta-mapping, which we use to devise an algorithm to *infer* meta-mappings from schema mappings with an approach that extends previous efforts for the definition of schema mappings by example; this algorithm is used to populate a repository of meta-mappings supporting schema mapping reuse (Section 3).
- An approach to reuse based on: (i) the search, in the repository of available meta-mappings those that *fit* a new pair of source and target schemas and (ii) the construction, from the retrieved meta-mappings, of possible mappings to be proposed to the designer (Section 4).
- Given the binary nature of fitness and the complexity of testing it, we introduce *coverage*, a feature-based metric based on nearest neighbour search that acts as a predictor for fitness; we show that coverage enables an efficient search in the repository for meta-mappings that best fit a given pair of schemas, as well as a sound ranking of the retrieved meta-mappings according to their "fitness distance" from the schemas under consideration (Section 5).

We provide a full-scale evaluation of the algorithms in our system with more than 20,000 real-world data transformations over 40,000 schemas. The results show that our solution is effective in practice and the optimizations enable an interactive use of the system (Section 6).

Because of space limitation, proofs for all theorems are in the full version of the paper [4].

2. BACKGROUND AND GOAL

In this section we recall the notion of schema mapping [15] and introduce that of *meta-mapping*. While the former notion is used to model specific data trations, the latter introduces an abstraction over mappings [23, 29] and allows us to model *generic* mappings between schemas. Building on these notions, we also clarify the goals of this paper.

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, $M \subseteq Inst(\mathbf{S}) \times Inst(\mathbf{T})$ [9].

Without loss of generality, 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. In this case, J is called a *solution* of I under σ. We can compute a suitable J in polynomial time by applying the *chase procedure* to I using σ [15]: the result may have labeled nulls denoting unknown values and is called the *universal* solution, since it has a homomorphism to any possible solution J', that is, a mapping h of the nulls into constants and nulls such that $h(J) \subseteq J'$.

EXAMPLE 4. *Consider the schemas $\mathbf{S}_A$, $\mathbf{S}_B$ and $\mathbf{G}$ of Figure 1, which we recall in Figure 3 with all the formalisms that will be discussed throughout the paper.*

A mapping between $\mathbf{S}_A$ and $\mathbf{G}$ is the st-tgd σ_A discussed in the Introduction and reported in Figure 3 (quantifiers are omitted for the sake of readability). Intuitively, the application of the chase to the instance of $\mathbf{S}_A$ using σ_A enforces 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 includes the first two tuples in relation `Balance` *in Figure 1.*

Besides, a mapping from $\mathbf{S}_B$ to $\mathbf{G}$ is represented by the s-t tgd σ_B in Figure 3. The result of the chase is a relation with the third and fourth tuple of relation `Balance`*.*

Specifying a mapping through st-tgds is a consolidated practice, as they offer a clear mechanism to define a relationship between the source and the target schema. Since we consider one dependency at a time, in the following we blur the distinction between mapping and dependency and simply refer to "the mapping". Moreover, we will call a set of mappings $\mathscr{H} = \{\sigma_1, \ldots, \sigma_n\}$ a *transformation scenario*.

2.2 Meta-mappings

A *meta-mapping* describes generic mappings between relational schemas and is defined as a mapping over the catalog of a relational database [29]. Specifically, in a relational meta-mapping, source and target are both defined over the following schema, called *(relational) dictionary*: $\mathsf{Rel}(\underline{name})$, $\mathsf{Att}(\underline{name}, in)$, $\mathsf{Key}(\underline{name}, in)$, $\mathsf{FKey}(\underline{name}, in, refer)$ (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 5. *Figure 3 shows also the m-schemas of the schemas $\mathbf{S}_A$, $\mathbf{S}_B$, and $\mathbf{G}$ of the running example.*

We assume, hereinafter, 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 the elements of a source m-schema map to the elements of a target m-schema.

EXAMPLE 6. *Mapping σ_A of Example 4 can be expressed, at the dictionary level, by meta-mapping Σ_A in Figure 3. This st-tgd describes a generic transformation that takes two source relations R and S linked by a foreign key F and generates a target relation T obtained by joining R and S on F that includes: the key K_1 and the attributes A_1 and A_2 from relation R and the attribute A_3 from S.*

Similarly to schema mappings, given a source m-schema S and a meta-mapping $\mathscr{M}$, we compute a target m-schema T by applying the chase procedure to S using $\mathscr{M}$.

EXAMPLE 7. *The chase of the m-schema $\mathsf{S_A}$ using Σ_A, both in Figure 3, generates the following target m-schema where $\perp_R$ is a labelled null denoting a relation name.*

Rel

name
$\perp_R$

Key

name	in
Firm	$\perp_R$

Att

name	in
Gains	$\perp_R$
Zone	$\perp_R$
Description	$\perp_R$

This m-schema describes the relational schema:

$$R(\underline{Firm}, Gains, Zone, Description)$$

A meta-mapping operates at schema level rather than at data level and thus provides a means for describing generic transformations. There are subtleties that could arise from the chase procedure 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 *equality generating dependency* (egd) [15] ensuring that whenever two relations in the target have the same structure, then they coincide.

2.3 From meta-mappings to mappings

Given a source schema $\mathbf{S}$ and a meta-mapping Σ, not only is it possible to generate a target schema by using the chase, as shown in Example 7, but it is also possible to automatically obtain a schema mapping σ that represents the *specialization* of Σ for $\mathbf{S}$ and $\mathbf{T}$ [29]. The *schema to data exchange transformation* (SD transformation) generates from $\mathbf{S}$ and Σ a complete schema mapping made of $\mathbf{S}$, a target schema $\mathbf{T}$ (obtained by chasing the m-schema of $\mathbf{S}$ with the meta-mapping), and an s-t tgd σ between $\mathbf{S}$ and $\mathbf{T}$. The correspondences between LHS and RHS of σ are derived from the provenance information computed during the chase step, in the same fashion as the provenance computed over the source instance when chasing schema mappings [13].

EXAMPLE 8. *Consider again the scenario in Figure 3. If we apply the* SD *transformation to the schema $\mathbf{S}_A$ and the meta-mapping Σ_A, we obtain the target m-schema of Example 7 and the following mapping from $\mathbf{S}_A$ to $\perp_R$:*

$$\sigma : R_A(f, g, z, s), Activity(s, d) \rightarrow \perp_R(f, g, z, d).$$

Thus, we get back, up to a renaming of the target relation, the mapping σ_A in Figure 3 from which Σ_A originates.

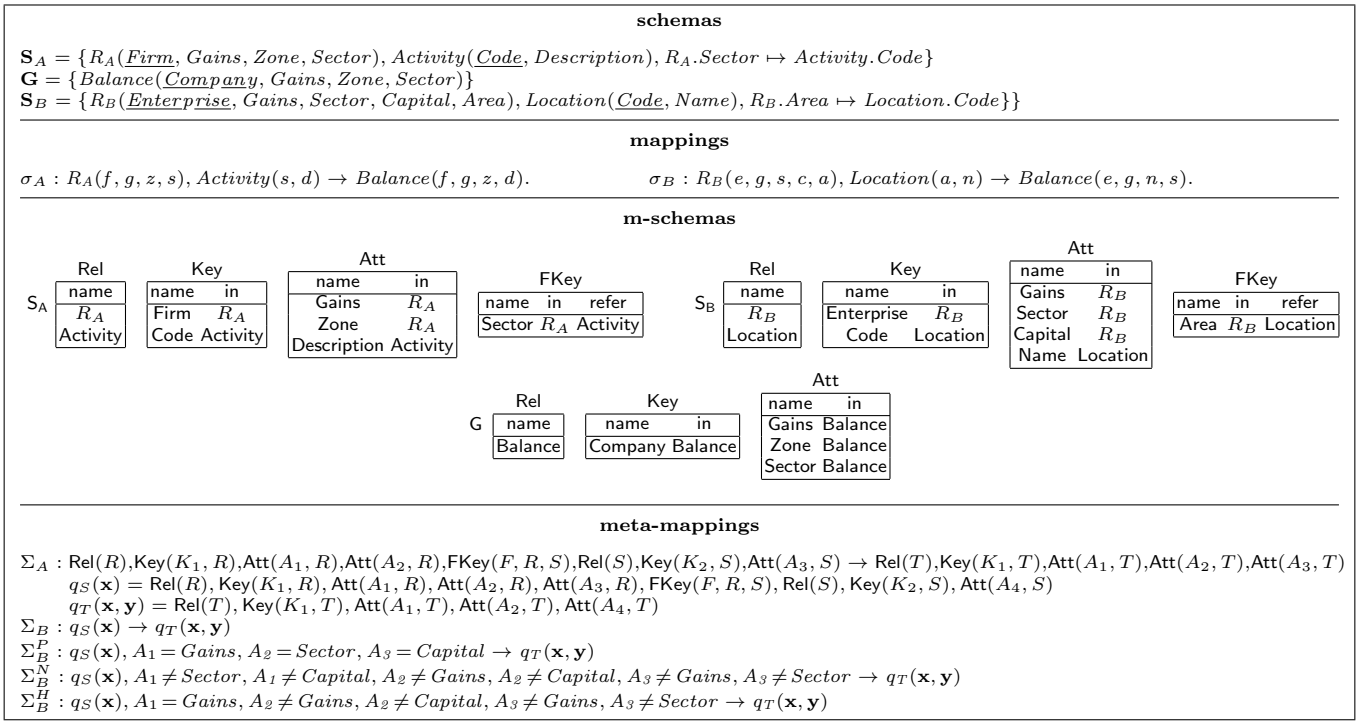

Figure 3: Schemas, m-schemas, mappings, and meta-mappings discussed along the paper.

2.4 Problem statement

While previous work focused on generating a schema mapping from a given meta-mapping [29], here we tackle the more general problem of *mapping reuse*, which consists of: (i) generating a repository of meta-mappings from a set of user-defined schema mappings, and (ii) given a new pair of source and target schemas, generating a suitable mapping for them from the repository of meta-mappings. For example, in the scenario of Figure 3, the goal is to reuse meta-mapping Σ_A, which has been inferred from σ_A between $\mathbf{S}_A$ and $\mathbf{G}$, to generate a mapping between $\mathbf{S}_B$ and $\mathbf{G}$.

3.FROM MAPPINGS TO META-MAPPINGS

In this section we describe how a schema mapping is generalized into a set of meta-mappings. We start by introducing the notion of fitness (Section 3.1), which formalizes the relationship between a meta-mapping and the mappings it generalizes. We then discuss how we infer fitting meta-mappings from mappings. This is done in two main steps: (1) the construction of a "canonical meta-mapping" (Section 3.2), which can be non-fitting, and (2) the "repair" of non-fitting canonical meta-mappings by means of suitable equality and inequality constraints (Section 3.3). The overall approach is then summarized in Section 3.4.

3.1 Fitness of meta-mappings

We say that there is a *matching* from S_1 to S_2 if there is an injective mapping μ from the elements of S_1 to the elements of S_2 such that: (i) μ preserves the identity on common constants (that is, for every constant c occurring in both S_1 and S_2, $\mu(c) = c$), (ii) μ maps nulls to constants, and (iii) when μ is extended to schemas, $\mu(\mathsf{S}_1) \subseteq \mathsf{S}_2$.

We are now ready for the definition of *fitness* of meta-mappings.

DEFINITION 1. *Let $\mathbf{S}$ and $\mathbf{T}$ be a pair of schemas and S and T be the m-schemas of $\mathbf{S}$ and $\mathbf{T}$, respectively. A meta-mapping Σ* fits *$\mathbf{S}$ and $\mathbf{T}$ if there is a matching between the universal solution of S under Σ and T. By extension, a meta-mapping Σ* fits *a mapping σ if it fits its source and target schemas.*

EXAMPLE 9. *To test if the meta-mapping Σ_A of our running example fits the mapping σ_A between $\mathbf{S}_A$ and $\mathbf{G}$ (both in Figure 3), we compute the universal solution of S_A under Σ_A by chasing S_A using Σ_A. The result is the following:*

T
Rel: name | $\perp_R$
Key: name in | Firm $\perp_R$
Att: name in | Gains $\perp_R$ | Zone $\perp_R$ | Description $\perp_R$

It is now easy to see that there is a matching from T to G (in Figure 3), which maps $\perp_R$ to Balance, Firm *to* Company, Description *to* Sector, *and leaves the other constants unchanged. It follows that, indeed, Σ_A* fits *σ_A.*

Note that the notion of matching is Boolean and, for simplicity, refers to the identity of names used in the schemas. However, we point out that this definition (and thus the notion of fitness) can be naturally extended by adopting a more general notion of similarity, such as those used in schema matching [8].

3.2 Canonical meta-mappings

We now consider a basic way to derive a fitting meta-mapping from a mapping σ, which we call *canonical*. It is

built by "projecting", in its source and target m-schemas the correspondences between attributes expressed by σ.

The *projection* of σ on its m-schemas S and T is a pair of m-schemas, denoted by S^σ and T^σ, obtained from S and T as follows: 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 "fresh" constant.

EXAMPLE 10. *The projections of σ_A in Figure 3 on its source and target m-schemas $\mathsf{S_A}$ and G are the following.*

$\mathsf{S_A}^{\sigma_A}$

Rel
name
R_A
Activity

Key	
name	in
c_f	R_A
Code	Activity

Att	
name	in
Gains	R_A
Zone	R_A
c_d	Activity

FKey		
name	in	refer
Sector	R_A	Activity

G^{σ_A}

Rel
name
Balance

Key	
name	in
c_f	Balance

Att	
name	in
Gains	Balance
Zone	Balance
c_d	Balance

Basically, constant c_f describes the fact that σ_A maps the key of relation R_A to the key of Balance*, whereas constant c_d describes the fact that a non-key attribute of* Activity *maps a non-key attribute of* Balance*. Note that the other attributes remain unchanged since they appear in both schemas.*

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 $\mathbf{S}$ to $\mathbf{T}$ is the 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 replaced by the same 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 not occuring in S replaced with the same existentially quantified variable in $\mathbf{y}$.*

EXAMPLE 11. *Consider again the mapping σ_A in Figure 3. From the projections of σ_A on its source and target m-schemas, presented in Example 10, we obtain precisely the canonical-meta mapping Σ_A (also in Figure 3). In Example 9 we have shown that, indeed, Σ_A fits σ_A.*

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 mapping it originates from.

EXAMPLE 12. *Let us consider the mapping σ_B in Figure 3. The projection of σ_B on its source and target m-schemas $\mathsf{S_B}$ and G are the following m-schemas.*

$\mathsf{S_B}^{\sigma_B}$

Rel
name
R_B
Location

Key	
name	in
c_e	R_B
Code	Location

Att	
name	in
Gains	R_B
Sector	R_B
Capital	R_B
c_n	Location

FKey		
name	in	refer
Area	R_B	Location

G^{σ_B}

Rel
name
Balance

Key	
name	in
c_e	Balance

Att	
name	in
Gains	Balance
c_n	Balance
Sector	Balance

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 $\mathsf{S_B}$ using Σ_B produce the following universal solution that does not match with G:

T

Rel
name
$\perp_R$

Key	
name	in
Enterprise	$\perp_R$

Att	
name	in
Gains	$\perp_R$
Sector	$\perp_R$
Capital	$\perp_R$
Name	$\perp_R$

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. We say that the *position* $\tau(q, x)$ of a variable x in a conjunctive formula q 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 containing x, and p_i is the relative position of x in a_i. For example, given: $q : \mathsf{Rel}(R), \mathsf{Att}(A, R)$ we have: $\tau(q, R) = \{(\mathsf{Rel}, 1), (\mathsf{Att}, 2)\}$.

DEFINITION 3. *Two distinct variables x_i and x_j are* potentially ambiguous *in a conjunctive formula q if $\tau(q, x_i) \cap \tau(q, x_j) \neq \emptyset$.*

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 recall that: a homomorphism h from $q(\mathbf{x})$ to I *extends to* another homomorphism h' from $q'(\mathbf{y})$ to J, if for every variable $x \in \mathbf{x} \cap \mathbf{y}$, $h(x) = h'(x)$.

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 13. *Variables A_1, A_2, A_3 of Σ_B in Figure 3 are ambiguous since: (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 Figure 3 are not ambiguous since they are potentially ambiguous in 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.

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

3.3 Repairing canonical meta-mappings

Taking inspiration from Theorem 1, which provides a necessary and sufficient condition to guarantee fitness, we can enforce fitness in canonical meta-mappings by including constraints that avoid the presence of ambiguous variables. More specifically, we extend a canonical meta-mapping by adding to its LHS a conjunction of *explicit* constraints of the form $x = c$ or $x \neq c$, in which x is a variable and c is a constant. We denote them as *positive*, *negative* or *hybrid* constraints if they involve equalities only, inequalities only or both, respectively.

EXAMPLE 14. *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 Figure 3, respectively. Possible extensions of Σ_B with constraints are reported below. They belong to different classes: positive (Σ_B^P), negative (Σ_B^N), and hybrid (Σ_B^H). All of them fit σ_B.*

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

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

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

Explicit constraints guarantee a form of semantic compliance between the meta-mapping and the scenario at hand. For instance, the meta-mappings in Example 14 are likely to be more suitable in application domains involving `Gains` and `Capital` rather than those involving `Students` and `Courses`. Negative constraints cover more cases but tend to be too general, while positive constraints are more restrictive but better capture a specific application domain. Therefore, to exploit the advantages of both, it is useful to generate also hybrid constraints, which involve both equalities and inequalities.

Algorithm 1 turns a non-fitting canonical meta-mapping $\Sigma = q_S \rightarrow q_T$ for a mapping σ between $\mathbf{S}$ and $\mathbf{T}$ into a fitting one by adding such constraints. We call such process a *repair* of the given meta-mapping.

This algorithm first computes the set $\mathscr{H}_S$ of all the possible homomorphisms from q_S to S^σ and the set $\mathscr{H}_T$ of all the possible homomorphisms from q_T to T^σ (lines 5-6). Then, we isolate the set $\mathscr{H}_E$ of homomorphisms in $\mathscr{H}_S$ that extend to some homomorphism in $\mathscr{H}_T$ (line 7). Lines 8-9 store 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 potentially ambiguous variables are indeed ambiguous. Whenever a homomorphism activates the potential ambiguity in the LHS (lines 12-13), if such ambiguity is not present in the RHS (line 14) or is present but, for some extension h', the potentially ambiguous variables occur in atoms that are mapped by h' to different facts (lines 15-16), then those variables are stored in $\mathscr{A}$ (line 17). The ambiguous variables in $\mathscr{A}$ are then fixed by means of the extended homomorphisms in $\mathscr{H}_E$. For this, we select the homomorphisms that assign different values to the ambiguous variables (line 19) and, for each of them, we build several repairs by adding a conjunction of constrains $\gamma_1, \ldots, \gamma_n$ (line 20) for each n-uple of ambiguous variables that appear in both q_S and q_T. Each constraint γ_i can be (line 21): (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.

EXAMPLE 15. *Assume we want to repair the canonical meta-mapping Σ_B in Example 8. Among the homomorphisms in $\mathscr{H}_E$, there are:*

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

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 such that: $h_i(A_1) \neq h_i(A_2) \neq h_i(A_3)$. Among them, the homomorphism h_1 above, from which we generate the mappings Σ_B^P, Σ_B^N and Σ_B^H in Figure 3.*

The following result shows the correctness of Algorithm 1.

THEOREM 2. *A meta-mapping that fits a given mapping σ always exists and Algorithm 1 always terminates and determines such meta-mapping.*

Unfortunately, the repair process unavoidably requires to detect all the ambiguities, a problem with inherent exponential complexity.

THEOREM 3. *The problem of enforcing fitness in a canonical meta-mapping is Π_2^p-hard.*

3.4 Meta-mapping inference

Algorithm 2 summarizes the overall technique to infer fitting meta-mappings from mappings. Given a schema mapping σ from $\mathbf{S}$ to $\mathbf{T}$, we first project σ on the m-schemas S and T of $\mathbf{S}$ and $\mathbf{T}$, respectively (line 5). We then build from S^σ and T^σ the canonical meta-mapping Σ (line 6), which may not fit σ. We finally repair Σ by adding explicit constraints, as discussed in Section 3.3, and produce a set $\mathscr{Q}$ of meta-mappings, all fitting σ (line 7). This method can be extended to a set $\mathscr{H}$ of mappings by selecting, among the meta-mappings generated, only those that fit all the mappings in $\mathscr{H}$.

The correctness of the overall approach follows by Theorem 2. In the worst case our technique requires exponential time, as an unavoidable consequence of the hardness of the underlying problem of repairing a canonical meta-mapping (Theorem 3). While the complexity is high, the algorithm is executed in a few seconds in practice with real schemas. Also, the inference can be done off-line, without affecting the users facing the system for mapping reuse.

Finally, note that there is the risk of generating a large number of different fitting variants out of a single non-fitting meta-mapping. We will show however in Section 6 that the presence of many variants increases the chances to find relevant solutions.

Algorithm 1 Meta-mapping repair.

procedure REPAIR-META-MAPPING
Input: a canonical meta-mapping $\Sigma = q_S \rightarrow q_T$ for a mapping σ between $\mathbf{S}$ and $\mathbf{T}$, S^σ and T^σ;
Output: a set $\mathscr{R}$ of meta-mappings *fitting* σ
  $\mathscr{R}, \mathscr{A} \leftarrow \emptyset$
  $\mathscr{H}_S \leftarrow$ generate all the homomorphisms from q_S to S^σ
  $\mathscr{H}_T \leftarrow$ generate all the homomorphisms from q_T to T^σ
  $\mathscr{H}_E \leftarrow \{h \in \mathscr{H}_S$ s.t. h *extends* to some $h' \in \mathscr{H}_T\}$
  $\Gamma \leftarrow \{(x_1, \ldots, x_n)$ 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 potential ambiguous variables in q_S
  $\Lambda \leftarrow \{(x_1, \ldots, x_n)$ 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 potential ambiguous variables in q_T
  for $h \in \mathscr{H}_E$ **do** ▷ here we detect ambiguous variables
    for $(x_1, \ldots, x_n) \in \Gamma$ **do**
      if $(a(h(k_1), \ldots, h(x_1), \ldots, h(k_m)) = \ldots = a(h(w_1), \ldots, h(x_n), \ldots, h(w_m))$
        for some $(a, \cdot) \in \bigcap_{i=1}^n \tau(q_S, x_i))$ **and** ▷ presence of an ambiguity in the LHS
        $((x_1, \ldots, x_n) \notin \Lambda$ **or**
          $(a(h'(k_1), \ldots, h'(x_1), \ldots, h'(k_n)) \neq \ldots \neq a(h'(w_1), \ldots, h'(x_n), \ldots, h'(w_n))$
          for some $(a, \cdot) \in \bigcap_{i=1}^n \tau(q_T, x_i)$ and some extension h' of $h))$ ▷ w/o a corresp. ambiguity in the RHS
      then $\mathscr{A} \leftarrow \mathscr{A} \cup \{(x_1, \ldots, x_n)\}$
  for $h \in \mathscr{H}_E$ **do** ▷ here we fix the ambiguities
    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
    then *add* $\forall \mathbf{x}(q_S(\mathbf{x}), \gamma \rightarrow \exists \mathbf{y} q_T(\mathbf{x}, \mathbf{y}))$ to $\mathscr{R}$ where:
      $\gamma = \bigwedge_{(x_1, \ldots, x_n) \in \mathscr{A}}(\gamma_1, \ldots, \gamma_n)$ and γ_i is either $x_i = h(x_i)$ or $\bigwedge_{i,j=1\ldots n,\ i \neq j} x_i \neq h(x_j)$
  return $\mathscr{R}$

Algorithm 2 Meta-mapping inference.

procedure INFER
Input: a mapping σ between $\mathbf{S}$ (whose m-schema is S) and $\mathbf{T}$ (whose m-schema is T)
Output: a set of *meta-mappings* $\mathscr{Q}$ fitting σ
  $\mathscr{Q} \leftarrow \emptyset$
  $(\mathsf{S}^\sigma, \mathsf{T}^\sigma) \leftarrow$ PROJECT-MAPPING$(\sigma, \mathsf{S}, \mathsf{T})$
  $\Sigma \leftarrow$ CANONICAL-META-MAPPING$(\mathsf{S}^\sigma, \mathsf{T}^\sigma)$
  $\mathscr{Q} \leftarrow$ REPAIR-META-MAPPING$(\Sigma, \mathsf{S}^\sigma, \mathsf{T}^\sigma)$
  return $\mathscr{Q}$

Algorithm 3 Mapping reuse.

1: **procedure** REUSE
2: **Input:** a set of meta-mappings $\mathscr{Q}$ and a pair of schemas $\mathbf{S}$ (with m-schema S) and $\mathbf{T}$ (with m-schema T)
3: **Output:** a set of schema mappings $\mathscr{S}$ from $\mathbf{S}$ to $\mathbf{T}$
4: $\mathscr{S}, \mathscr{M} \leftarrow \emptyset$
5: **for** $\Sigma \in \mathscr{Q}$ **do**
6: $\mathsf{U} \leftarrow$ CHASE(S, Σ)
7: $\mathscr{M} \leftarrow$ MATCHINGS(U, T)
8: **if** $\mathscr{M}$ is not empty **then**
9: $\sigma \leftarrow$ SD-TRANSFORMATION(S, Σ)
10: **for** $\gamma \in \mathscr{M}$ **do** $\mathscr{S} \leftarrow \mathscr{S} \cup \gamma(\sigma)$
11: **return** $\mathscr{S}$

4. MAPPING REUSE

In this section we first present an approach to the reuse of schema mapping that relies on the availability of a repository of meta-mappings built as shown in the previous section (Section 4.1) and then discuss how to implement this approach in an efficient and robust way (Section 4.2).

4.1 An approach to reuse based on fitness

The first, fundamental, step towards the reuse of a set of mappings $\mathscr{H}$ is the construction of a repository of meta-mappings $\mathscr{Q}$ inferred from $\mathscr{H}$ using Algorithm 2. At this point, if we need to map a new source schema $\mathbf{S}$ to a new target schema $\mathbf{T}$, we can reuse the knowledge in the original mappings $\mathscr{H}$ by searching any meta-mapping Σ in $\mathscr{Q}$ that fits $\mathbf{S}$ and $\mathbf{T}$ and generate from it a mapping from $\mathbf{S}$ to $\mathbf{T}$. The rationale under this approach is the following: if Σ fits $\mathbf{S}$ and $\mathbf{T}$ in the same way in which it fits, by construction, the source and the target of the mapping σ from which it has been inferred, then it will generate a mapping from $\mathbf{S}$ to $\mathbf{T}$ that "mimic" σ on these schemas.

Thus, mapping reuse can proceed as described by Algorithm 3. This algorithm takes as input a repository of meta-mappings $\mathscr{Q}$ (built from an initial set of mappings $\mathscr{H}$ using Algorithm 2) and a pair of schemas $\mathbf{S}$ and $\mathbf{T}$, and returns a set of possible mappings between $\mathbf{S}$ and $\mathbf{T}$.

For each meta-mapping Σ in $\mathscr{Q}$ (line 5) we first test if Σ fits $\mathbf{S}$ and $\mathbf{T}$: according to Definition 1, this can be done by first computing the universal solution U of S under Σ by chasing S with Σ (line 6) and then checking for the existence of matchings from U to T (line 7). If Σ fits $\mathbf{S}$ and $\mathbf{T}$ (line 8), we generate from Σ a mapping σ from $\mathbf{S}$ to $\mathbf{T}$ using the SD transformation discussed in Section 2.3 (line 9) and, for each matching γ between U and T, we add $\gamma(\sigma)$ to the set of possible solutions (line 10).

EXAMPLE 16. *Consider the possible reuse of the mapping σ_B between $\mathbf{S}_B$ and $\mathbf{G}$ in Figure 3 for generating a mapping between $\mathbf{S}_A$ and $\mathbf{G}$. As discussed in Example 15, the meta-mappings that can be inferred from σ_B include Σ_B^P, Σ_B^N and Σ_B^H, reported in Figure 3. Consider now Σ_B^H.*

By chasing the m-schema $\mathsf{S_A}$ of $\mathbf{S}_A$ using Σ_B^H, we obtain the following universal solution of $\mathsf{S_A}$ under Σ_B^H:

U

Rel
name
$\perp_R$

Key	
name	in
Firm	$\perp_R$

Att

name	in
Gains	$\perp_R$
Zone	$\perp_R$
Description	$\perp_R$

Table 1: Applicability of meta-mappings.

	$\mathbf{S}_A, \mathbf{G}$	$\mathbf{S}_B, \mathbf{G}$	$\mathbf{S}_C, \mathbf{G}$	$\mathbf{S}_A, \mathbf{G}'$	$\mathbf{S}_D, \mathbf{S}_E$
Σ_A	✓	×	×	✓	×
Σ_B	✓	×	×	✓	×
Σ_B^P	×	✓	×	×	×
Σ_B^N	✓	✓	✓	✓	×
Σ_B^H	✓	✓	✓	✓	×
Σ_B^O	×	✓	×	×	×

It is easy to see that there is a matching γ from U *to the m-schema* G *of* $\mathbf{G}$ *that maps:* $\perp_B$ *to* Balance, Firm *to* Company, *and* Description *to* Sector. *If we apply the* SD *transformation to* $\mathsf{S_A}$ *and* Σ_B^H *we obtain the following mapping* σ*:*

$$\sigma : R_A(f, g, z, s), Activity(s, d) \rightarrow \perp_R(f, g, z, d).$$

Hence, we can finally return $\gamma(\sigma) = \sigma_A$ *to the user as a possible suggestion for a mapping between* $\mathbf{S}_A$ *and* $\mathbf{G}$.

Consider now Table 1, where the columns denote meta-mappings and the rows denote pairs of source and target schemas: it shows with a ✓ if it is possible to generate, using the process described above, a suitable mapping between a pair of schemas from a given meta-mapping. Using this table, we now discuss various scenarios to better explain how our definition of mapping reuse works in practice.

New target schema. Consider the case when we have a new target schema, which is a modified version of $\mathbf{G}$:

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

where one attribute has been changed from *Zone* to *Area*. The meta-mapping Σ_B^H can still be applied on $\mathsf{S_A}$ producing the following target schema:

$$\mathbf{T} = \{Balance(\underline{Firm}, Gains, Zone, Description)\}.$$

Notice that now there are two possible isomorphisms between T and G: one mapping *Zone* to *Area* and *Description* to *Sector* (correct), and a second one mapping *Zone* to *Sector* and *Description* to *Area* (incorrect). In these cases, we expose both mappings and let the user decide.

No reusable meta-mappings. Consider now a new scenario, which is not structurally related to the bank scenarios and it is defined as follows:

$$\mathbf{S}_D = \{R_1(\underline{A_1}, A_2, A_3), R_2(\underline{A_1}, A_4), R_1.A_1 \mapsto R_2.A_1\}$$
$$\mathbf{S}_E = \{R_3(\underline{A_5}, A_1, A_3))\}$$

No matter what are the labels for attributes and relations, there is no meta-mapping in our examples that consider the case where the foreign key is defined over two keys. As no meta-mapping with this structure has been defined yet, the search would result empty for this pair of schemas.

Overfitting meta-mappings. Finally, in addition to the meta-mappings that we have already introduced, we consider a meta-mapping Σ_B^O that fully specifies the attribute names which is defined as follows:

$$\Sigma_B^O : q_S(\mathbf{x}), R = R_B, A_1 = Gains, A_2 = Sector,$$
$$A_3 = Capital, K_1 = Enterprise, F = Area,$$
$$S = Location, K_2 = Code, A_4 = Description \rightarrow q_T(\mathbf{x}, \mathbf{y})$$

where $q_S(\mathbf{x})$ and $q_T(\mathbf{x}, \mathbf{y})$ are those in Figure 3.

Interestingly, it turns out that the hybrid meta-mappings Σ_B^H appears to be the most usable, since it is not applicable only to the pair $\mathbf{S}_D$ and $\mathbf{S}_E$, which belong to a different scenario. On the other extreme, the meta-mapping Σ_O applies only to $\mathbf{S}_B$ and $\mathbf{G}$, as it is built from an exact correspondence to schema elements of σ_B and works only for this case.

The examples suggest that the mechanism for inferring meta-mappings illustrated in this section provides a powerful tool for mapping reuse.

4.2 Practical approach to reuse

The reuse technique illustrated in Section 4.1 is solid but the search for fitting meta-mappings in the repository, as done in Algorithm 3, has two drawbacks, as follows.

1. It is Boolean: a pair of schemas either fits a meta-mapping or not. Hence, the search returns empty results even if there are meta-mappings that need only minor adjustments to fit the input schemas. For instance, Σ_B^P in Figure 3 does not fit schemas $\mathbf{S}_A$ and $\mathbf{G}'$ (Table 1), yet Σ_B^P is certainly better than Σ_O (Section 4.1). In fact, it is sufficient to drop the constraint $A_1 = Gains$ from it to fit $\mathbf{S}_A$ and $\mathbf{G}'$; conversely, Σ_O would require multiple changes to fit those schemas.

2. It requires to scan through all the meta-mappings in the repository and check whether each of them fits a given pair of schemas, a task that is computationally expensive: it is polynomial in the number of meta-mappings, and, for the single check, exponential in the number of schema elements.

To overcome these limitations, we introduce the notion of *coverage*, a parametric distance metric that, using a machine learning technique based on *nearest neighbor search* with *supervised distance metric learning*, works as a predictor for fitting meta-mappings. Specifically, given $\mathbf{S}$ and $\mathbf{T}$, coverage ranks as top results the meta-mappings that, based on their structure and on constants involved, most likely fit $\mathbf{S}$ and $\mathbf{T}$. This solution not only enables efficient retrieval, but it also ranks meta-mappings according to the amount of change needed to make them usable for the given schemas.

The approach relies on a set of features that are learnt with a supervised technique and is based on three phases:

- a *feature engineering phase*, in which some features are identified to describe meta-mappings and schemas: number of relations, number and kind of joins, number and kind of constraints, constant values;

- a *supervised distance metric learning phase*, in which a set of schemas and meta-mapping pairs labeled as fitting/non fitting are used to learn the coverage, which scores the level of fitness of a meta-mapping for a pair of schemas;

- a *test phase*, where the coverage is applied on an input pair of source and target schema and ranks the meta-mappings that best fit the given scenario.

The coverage metric enables top-k ranking of the most suitable meta-mappings, thus addressing problem 1 above. Also, as the fitness prediction takes constant time, the test phase takes linear time in the worst case, thus addressing problem 2. Furthermore, a distance metric allows the use of data structures for fast retrieval of top-ranked results and we exploit this property to design a solution that operates in LOGTIME. We detail this solution in the next section.

5. SEARCHING THE REPOSITORY

We now present in detail *coverage*, our distance metric between meta-mappings and schemas (Section 5.1). We then describe how we use coverage to perform efficiently the search for meta-mappings over a repository (Section 5.2).

5.1 The Coverage distance-metric

The coverage is a distance metric that enables a comparison between the LHS and the RHS of a meta-mapping Σ with a pair $\mathbf{S}$ and $\mathbf{T}$ of source and a target schemas, respectively. If such distance is low, then it is likely that Σ fits $\mathbf{S}$ and $\mathbf{T}$. In addition, coverage can be used to compare the LHS and RHS of two meta-mappings, which allows us to organize and index the repository of meta-mappings, with the goal of reducing the search time for a given pair of schemas. In the following, we discuss the case of comparing the LHS of a meta-mapping Σ with a source schema $\mathbf{S}$, as the other case is similar.

From Fitness to Coverage. Recalling Definition 1 (fitness), we have that Σ fits $\mathbf{S}$ and $\mathbf{T}$ if there is a matching from the universal solution U of S under Σ to T (where S and T are the m-schemas of $\mathbf{S}$ and $\mathbf{T}$, respectively). While proper fitness holds only if there exists a mapping from U to T that preserves all constants, a predictor of fitness is a distance metric that just counts the number of constants shared by U and T. We call this measure *domain distance*. Here, the constants act as *domain features*, representing the domain of interest of a specific meta-mapping or schema. Similarly, the presence of a matching from U to T denotes that the structure of U is preserved; still, we can predict how much structure is actually shared. This is done by classifying the schema elements into *structural features* and using a *structural distance* between the sets to compare them.

Domain Distance. The *domain distance* δ measures the distance along a domain feature, i.e., it scores how likely the LHS of Σ and $\mathbf{S}$ deal with the same domain based on the presence of shared constants. A constant is shared if an atom of the meta-mapping is bound to an element of the schema. For example, this happens if Σ involves an atom $\mathsf{Att}(A, R)$ together with a constraint $A = \textit{Gains}$ and *Gains* is an attribute of $\mathbf{S}$. The same applies to relation names, keys, and foreign keys. More specifically, we adopt the *Jaccard distance* between the sets of constants:

$$\delta_{\Sigma,\mathbf{S}} = 1 - \frac{\mid \text{constants}(\Sigma) \cap \text{constants}(\mathbf{S}) \mid}{\mid \text{constants}(\Sigma) \cup \text{constants}(\mathbf{S}) \mid}$$

where we consider the set of all the constant names used in the meta-mapping or in the schema.

Structural Distance. The *structural distance* d measures the distance along a structural feature f_Σ of the LHS of Σ, and the corresponding one $f_\mathbf{S}$ for $\mathbf{S}$. Structural 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. Some of them specifically refer to the LHS of the meta-mapping, interpreted as the source and target schemas, respectively, when applied to schemas. An exhaustive list of features is in the full version of the paper [4]. To combine the feature, we adopt a $[0, 1]$-normalized distance of cardinalities:

$$d_{f_\Sigma, f_\mathbf{S}} = \frac{\mid f_\Sigma - f_\mathbf{S} \mid}{\max(f_\Sigma, f_\mathbf{S})}.$$

Coverage. The *coverage* χ is then defined as follows.

$$\chi = \frac{\prod_{i=1}^{n}(1 - d_{f^i_\Sigma, f^i_\mathbf{S}}) \times (1 - \delta_{\Sigma,\mathbf{S}})}{\prod_{i=1}^{n}(1 - d_{f^i_\Sigma, f^i_\mathbf{S}}) \times (1 - \delta_{\Sigma,S}) + \delta_{\Sigma,S} \times \prod_{i=1}^{n}(d_{f^i_\Sigma, f^i_\mathbf{S}})}$$

For every pair of common features f^i_Σ and $f^i_\mathbf{S}$ of Σ and $\mathbf{S}$, we consider their *score* $s = (1 - d_{f^i_\Sigma, f^i_\mathbf{S}})$, which indicates the probability that Σ covers $\mathbf{S}$ according to those features. A score of 0.5 denotes that a feature is not informative to predict fitness. A score lower than 0.5 indicates that the meta-mapping is likely to fit the schema with many modifications, while a higher score indicates that the meta-mapping is likely to fit the schema with few modifications. The *coverage* is actually built as the *Graham combination* [19] of the scores for all features. Moreover, our model is enhanced with a parametric scaling of the features, which accounts for the relative importance of each of them. This is obtained with the application of a parametric scaling formula to each score s, which adjusts the positive and negative contribution of s by altering the range of s. Thus, all scores are in the parametric range $[a, b]$, and so in χ each score s is replaced with $a + s \times (b - a)$. Assuming that a score of 0.5 is not informative, a and b behave as follows: (i) $[0.5, 0.5]$: s_f is neutral, (ii) $[0.5, >0.5]$: s_f has only a positive effect, (iii) $[<0.5, 0.5]$: s_f has only a negative effect, (iv) $[<0.5, >0.5]$: s_f has both a positive and a negative effect.

Classifier Training. As our distance metric is parametric in the ranges, we adopt a supervised training phase to learn a and b for each feature. This can be done in many ways with standard ML techniques. In particular, we train a *logistic classifier* [10] with a training set consisting of pairs $\langle \mathbf{X}, Y \rangle$, where $\mathbf{X}$ is a vector of scores for each sample and Y is a target variable denoting whether we are in a fitting situation (hence the value is one), or not (value zero). For each sample, the following applies:

$$\text{logit(p)} = \mathbf{X}\beta$$

with $\text{logit}(p) = \ln \frac{p}{1-p}$, p is the fitness probability, and β a vector of the b parameters in the ranges $[a, b]$ of the features. Solving the formula w.r.t. β, we obtain, for each feature f, its logarithmic odds, i.e., the marginal relevance of f, from which we derive $b = \frac{e^{\beta_f}}{1+e^{\beta_f}}$. The same is done to calculate parameter a. Values a and b of each feature are then used to evaluate χ. The estimated parameters can be directly used in the so-trained logistic classifier for binary fitness estimation. We experimentally show that effective configurations are identified with few training examples.

We point out that there are cases where the coverage does not return the same result of a fitness test. For example, given $\mathbf{S}_A$ and $\mathbf{G}$, the meta-mappings Σ^N_B and Σ^H_B are fitting, but Σ^P_B is not. However, they have identical scores based on the proposed features, as they have the same structure and the same intersection of attribute labels (Sector and Gains).

5.2 Searching meta-mappings

Thanks to the notion of coverage, we are now able to perform nearest neighbor search (NNS) and find the top-k meta-mappings in the repository that are likely to fit a given a pair of source and target schemas. While in principle NNS requires an exhaustive comparison of the pair of schemas with all the available meta-mappings, we propose

an optimized approach, based on the construction of a *meta-mapping index*, which allows us to efficiently find the top-k meta-mappings for the given schemas according to NNS.

Specifically, for each meta-mapping Σ in the repository, we compute the vector of its structural features f_Σ and use it to position the meta-mapping in a multi-dimensional space, which we with a *k-d tree* [7], a space-partitioning data structure that is built in $O(n \log n)$ and enables to retrieve points in $O(\log n)$. As distance function, we use $1 - \chi_S$, where χ_S is the coverage between two meta-mappings as defined in Section 5.1. To strike a trade-off between granularity and efficiency, we use only structural features in the index.

With the index in place, given a pair of input schemas, we can: (i) compute their features and build the corresponding vector; (ii) look up such vector in the meta-mapping index to obtain all meta-mappings that are close to the input; (iii) calculate the coverage only for these meta-mappings.

6. EXPERIMENTAL EVALUATION

We have evaluated our approach on real-world transformation scenarios using our system, GAIA, which implements the methods illustrated in the previous sections. The evaluation setting is described in Section 6.1. In Section 6.2, we report on the efficiency of GAIA in terms of inference and search time. In Section 6.3, we show the effectiveness of the coverage in ranking results, while in Sections 6.4 and 6.5, we report on the quality of the returned transformations in large scale experiments. Finally, we compare our approach against a repository of schema mappings in Section 6.6.

6.1 Experimental setup

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

Datasets and Transformations. We used a real-world scenario crafted from the data transformations that are periodically done to store data coming from several data 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 Table 2. In particular, the Chamber of Commerce provides data for 20,000 companies (datasets **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 between relations involving up to 30 attributes. Data for 20,000 more companies are collected by a commercial data provider in a single database (**CDP**) and then imported into CNBS. This database is different both in structure and attributes names from the schemas in **Chamber** and requires one mapping involving 15 joins in its LHS. Finally, data for further 1,300 companies (**Stock**) is imported from the stock exchange into CNBS, with each company requiring a mapping with 40 joins on average.

Transformation scenarios. For the inference of the meta-mappings, we have considered two configurations: *single*, where a meta-mapping is inferred from one mapping, and *multiple*, where the inference is computed on a group of mappings 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.

Table 2: Statistics on mappings and schemas.

Dataset	Total # st-tgds	Avg # rels	Total # atts	Avg # atoms
Chamber	20,000	5	900	32
CDP	1	23	800	15
Stock	1,300	34	900	40

6.2 Inference and search times

In Figure 4 (left), we report the times for computing the inference of meta-mappings with Algorithm 2, which relies on the notion of fitness. We infer meta-mappings from original mappings with an increasing number of ambiguous variables (x-axis: 2-10), hence requiring the evaluation of an exponentially growing number of homomorphisms. With 20,000 mappings and 10 ambiguous variables, more than 20 millions of meta-mappings are generated in about a minute.

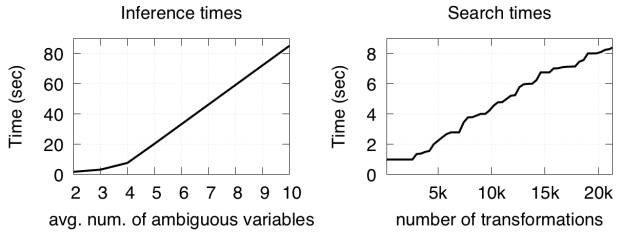

Figure 4: Inference and search execution times.

For the largest repository, we then report in Figure 4 (right) the time to execute the search procedure based on coverage with the k-d tree. 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.

6.3 Coverage and fitness

We now show with a set of meta-mappings labelled as fit/unfit that the coverage effectively ranks at the top the fitting meta-mappings for a given scenario.

Consider 4 pairs of source and target schemas (($\mathbf{S}_A, \mathbf{G}$), ($\mathbf{S}_B, \mathbf{G}$), ($\mathbf{S}_C, \mathbf{G}$), ($\mathbf{S}_D, \mathbf{G}$)) and a repository with 5 meta-mappings ($\Sigma_A, \Sigma_B, \Sigma_B^P, \Sigma_B^N, \Sigma_B^H$) discussed in Table 1. For each pair, we retrieve from the repository a list of meta-mappings ranked according to their coverage for the given scenario. We compute the feature scores for this experiment by using a training set of 20 examples (we discuss later the impact of the number of examples).

Table 3 reports, for each pair of source and target schemas in the training set: (i) the number of fitting meta-mapping in the repository and (ii) the precision in the top-k results ($1 \leq k \leq 4$) ranked according to the coverage measure.

The coverage ranks at the top the fitting meta-mappings, when they are in the repository for the given scenario, with perfect results for ($\mathbf{S}_B, \mathbf{G}$) and ($\mathbf{S}_C, \mathbf{G}$). For ($\mathbf{S}_A, \mathbf{G}$), a

Table 3: Precision@k according to coverage.

Scenario	# of fitting m-m	k=1	k=2	k=3	k=4
$\mathbf{S}_A, \mathbf{G}$	4/5	1.0	0.75	0.75	0.75
$\mathbf{S}_B, \mathbf{G}$	3/5	1.0	1.0	1.0	0.75
$\mathbf{S}_C, \mathbf{G}$	2/5	1.0	1.0	0.67	0.5
$\mathbf{S}_D, \mathbf{E}$	0/5	0	0	0	0

fitting meta-mapping is retrieved at k=1, but three meta-mappings having the same score follow it in the ranking (Σ_B^P, Σ_B^N and Σ_B^H). Since among these three there is one that is not fitting (Σ_B^P), the accuracy is 0.75 for k=2, 3, 4. In fact, the non fitting meta-mapping can be identified only with expensive homomorphism checks. Finally, for ($\mathbf{S}_D, \mathbf{E}$) the precision is always zero because there are no fitting meta-mappings in the repository. Clearly, this reflects the quality of the repository w.r.t. σ_D rather than the quality of the coverage in ranking meta-mappings.

Notice that the coverage comes with a *measure* of fitness. This is crucial for two reasons. First, it identifies meta-mappings that are close to fit a given scenario. For instance, while Σ_B^P does not fit ($\mathbf{S}_A, \mathbf{G}$), its coverage is 0.49, as a change in a constant would make it fitting. Conversely, no meta-mapping in the repository fits ($\mathbf{S}_D$, $\mathbf{E}$) and the coverage is always lower than 0.39. Second, the score allows us to take a binary decision on fitness based on a threshold. By manually setting a threshold of 0.48, the coverage correctly identifies all the fitting meta-mappings in the experiment except one, for a precision in classification of 0.95.

We evaluated the impact of small training datasets over test data. The classification precision varies between 0.90 and 0.95 in a five-fold cross validation with 4 training examples. It turns out that, with little effort in the tuning of the feature scores, the coverage correctly distinguishes fitting and non fitting meta-mappings (detailed results are in [4]).

The experiments with labeled scenarios show that coverage is an effective measure for ranking fitting meta-mappings. As labeling meta-mappings for fitness is an expensive exercise, in the following large-scale experiments we use the coverage to evaluate the quality of the retrieved meta-mappings.

6.4 Search precision

Let σ be a mapping from a source $\mathbf{S}$ to a target $\mathbf{T}$ and let $\mathscr{Q}$ be the set of meta-mappings inferred from σ. We now measure, in terms of *search precision*, the ability of the system to return the meta-mappings in $\mathscr{Q}$ when queried with $\mathbf{S}$ and $\mathbf{T}$, i.e., how well the system retrieves correct cases.

We use a *resubstitution* approach: we populate an initially empty repository by inferring the meta-mappings from a set Θ of mappings. For each schema mapping σ in Θ, we will denote by $\mathscr{Q}_\sigma$ all the meta-mappings inferred from σ. We then pick a schema mapping σ' from Θ and search the repository by providing as input the source and target schemas of σ'. We then collect the top-10 results according to the coverage and compute the percentage of correctly retrieved meta-mapping, that is, those that belong to $\mathscr{Q}_{\sigma'}$.

We test search precision with a corpus of mappings of increasing size (from 200 to 21,301). In each test, we query the repository of meta-mappings inferred from the given mappings by using 50 different source-target pairs and report their average search precision. The experiment is executed on single and multiple configurations. The largest repository contains about 700K explicit meta-mappings in single configuration and 110K in multiple configuration. 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. On average, a meta-mapping includes 12 equality and/or inequality constraints.

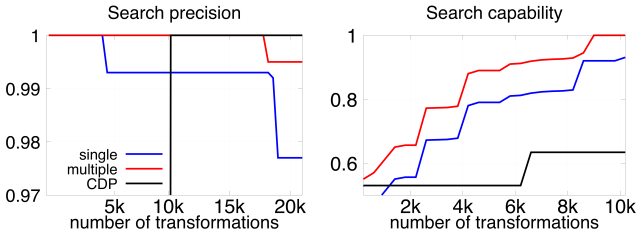

Figure 5: Search precision and search capability.

The chart on the left of Figure 5 shows that search precision decreases with the size of the repository as larger numbers of mappings lead to an increasing number of false positives in the result. We report the search precision for the **CDP** mapping with a separate line. As soon as this mapping is inserted in the repository (after about 10,200 transformations), it is immediately identified by the search with all configurations. This shows that specific structures and constants lead to meta-mappings that are easy to retrieve.

Inferring meta-mappings from multiple schema mappings improves search precision for two main reasons. First, the meta-mappings are inferred from a larger number of similar cases and are therefore more general, reducing the risk of overfitting. Second, the size of the repository decreases, which reduces the number of false positive.

6.5 Search capability

We now evaluate GAIA in the search of valuable transformations for completely new schemas, not seen by the system yet. For this task, we introduce the notion of *search capability*. Given a mapping σ from $\mathbf{S}$ to $\mathbf{T}$ and the set of meta-mappings $\mathscr{Q}_\sigma$ inferred from σ, we search suitable meta-mappings for $\mathbf{S}$ and $\mathbf{T}$ in the repository, making sure that it does not include any meta-mapping in $\mathscr{Q}_\sigma$. We then measure the search capability by comparing the k best retrieved meta-mappings and the meta-mappings in $\mathscr{Q}_\sigma$. This is done by using the coverage between meta-mappings (Section 5.1).

In the experiment, we resort to a *hold-one-out* technique. We start from the empty repository of meta-mappings 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 $\sigma \in \Phi$, we query the repository giving their source and target schemas 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 capability on the size of the corpus of mappings and of the transformation scenario. For each test run, the repository is populated with the meta-mappings generated by a number of mappings (from 200 to 10,200) in which we do not include **CDP**, in order to assess how well the system provides mappings for it. We then compute the search capability with 50 different source-target pairs, including also the source and the target of **CDP**.

As apparent in Figure 5 (right) the more meta-mappings are added to the repository, the higher is the search capabil-

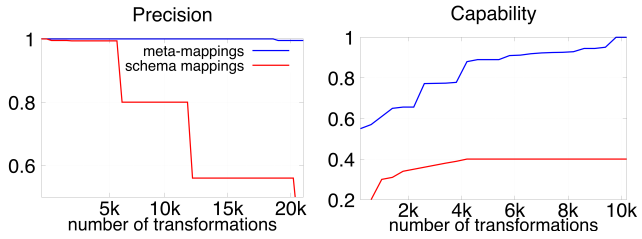

Figure 6: Precision and capability using a repository of mappings and a repository of meta-mappings.

ity. This is expected, as the availability of a large repository increases the likelihood to find a suitable meta-mapping for the given input. For **CDP**, we report only the best result, which is obtained with a multiple configuration. We observe a significant improvement after the inclusion of the transformation number "6400", which is closer to the size of **CDP**.

Grouping mappings as a single transformation scenario (multiple configuration) improves performance also in this case. Not only is the size of the repository reduced, but for the same number of input mappings, we observe an increased search capability for all configurations. This shows that there is an advantage in including in the repository meta-mappings that fits not only one, but several mappings.

These tests show that rather simple mappings can be derived very well with a sufficiently large set of meta-mappings. Conversely, very complex mappings can be hardly derived from meta-mappings that do not originate from them.

6.6 Meta-Mappings vs Schema Mappings

Finally, we repeat the experiments using the real-world datasets to measure search precision and search capability of our system when the repository is populated with mappings rather than meta-mappings. To build the repository, we compute the canonical meta-mapping for each mapping and bind all the variables to the specific names of attributes of source and target schemas (e.g., Σ_B^O in Section 4.1), turning them into pure mappings. We then compare the results with the case in which the repository contains meta-mappings.

In Figure 6, the search precision using a repository of schema mappings is comparable to the search precision using meta-mappings up to about 5,000 input mappings. Then the response becomes inaccurate because of the specificity of mappings, which lead to many false positives in the search. The differences between the two approaches is even more apparent if we consider the search capability. For new, unseen cases, a repository of mappings does not provide significant benefits, never going above 0.4 for any configuration. This demonstrates the limits of using schema mappings, that are hardly usable for similar but not identical cases.

7. RELATED WORK

While the notion of reuse is popular for software components [17], it got attention in the database community only recently [1]. The motivation comes from the increased capability of storing enterprise data and metadata in non-structured repositories, such as *data lakes* [16, 21, 31] and *knowledge graphs* [6], and from the wish to exploit "enterprise knowledge" by means of data management tasks. A recent example of the reuse approach is the automatic adaptation of rules for fraud detection across different datasets [28].

Our work follows this direction by introducing a novel framework for the reuse of schema mappings. We adopt the standard language of tgds for the declarative definition of schema mappings [15], enriched with the semantics of meta-mappings for the problem of exchanging metadata instead of data [29]. One contribution in data exchange is the semi-automatic generation of schema mappings [14, 20] given simple correspondences between the schema elements [5, 8]. Matching discovery support data transformation design [24], but it must be done for each new pair of schemas. Also, schema mappings are usually manually tuned with extra information coming from the user background knowledge [3]. Our system is able to reuse this manual refinement. There exist proposals to store *element matches* for reuse [26], but they cannot store logical formulas. Therefore, crucial information, such as the structure of the schema and the manual tuning, is completely lost, as experimetally demonstrated in Section 6.6. There is also a previous effort on mapping reuse, based on the inference of types in schemas [32]. This approach heavily relies on names (and possibly synonyms), while ours adopts a combination of names and structure, with a much wider potential for reuse.

Our inference of meta-mappings from mappings can be seen as the lifting to meta-level of the discovery of mappings from data examples [2, 3, 11, 18, 30], which in turn is inspired by the discovery of queries given data examples [27]. In our setting, previous algorithms [2, 3] would generate meta-mappings that do not capture the semantics of a transformation. Another theoretical framework for the discovery of schema mappings addresses unsuitable mappings by examining different repairs and solving a cost-based optimization problem to choose the best ones [18]. These results are not directly applicable to our context for two reasons: they operate at data level, and we have shown that plain schema mappings fail to generalize (see Section 6.6); then, they contribute intractability results, which reinforce our need for heuristic search techniques. The latter framework has also been extended with polynomial-time approximation algorithms to quickly choose the best repairs [30]. This is useful in practice, yet would be limiting in our setting, where we are actually interested in storing all the variants to maximise reuse.

ETL tools and research and commercial systems for *data wrangling* [22, 25] allow to store, load and combine previously defined transformations without any schema-level generalization or indexing of the transformation semantics. As shown in the experiments, our inference algorithm increases the chances of reusing previous mappings.

8. CONCLUSIONS

We introduced a system to support the design of schema mappings. Starting from a set of schema mappings, we generate more generic meta-mappings, which capture the semantics of the input mappings 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 then provide a list of "suitable" schema mappings. Experiments confirm the high quality of the retrieved mappings.

One direction for future work is the extension of the approach to schema mappings that include constants. This would allow us to capture and reuse data-level constraints that are meaningful to the user.

9. REFERENCES

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

[2] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23, 2011.

[3] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. Designing and refining schema mappings via data examples. In *SIGMOD*, pages 133–144, 2011.

[4] P. Atzeni, L. Bellomarini, P. Papotti, and R. Torlone. Metamappings for schema mapping reuse. Full version, 2018. `http://www.eurecom.fr/~papotti/files/GaiaTR18.pdf`.

[5] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer, 2011.

[6] L. Bellomarini, G. Gottlob, A. Pieris, and E. Sallinger. Swift logic for big data and knowledge graphs. In *IJCAI*, pages 2–10. ijcai.org, 2017.

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

[8] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.

[9] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, 2007.

[10] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[11] A. Bonifati, U. Comignani, E. Coquery, and R. Thion. Interactive mapping specification with exemplar tuples. In *SIGMOD*, pages 667–682, 2017.

[12] C. Chen, B. Golshan, A. Y. Halevy, W. Tan, and A. Doan. Biggorilla: An open-source ecosystem for data preparation and integration. *IEEE Data Eng. Bull.*, 41(2):10–22, 2018.

[13] L. Chiticariu and W. C. Tan. Debugging schema mappings with routes. In *VLDB*, pages 79–90, 2006.

[14] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling*, 2009.

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

[16] R. C. Fernandez, Z. Abedjan, S. Madden, and M. Stonebraker. Towards large-scale data discovery. In *ExploreDB*, pages 3–5, 2016.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995.

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

[19] P. Graham. Better bayesian filtering. In *Proceedings of Spam Conference*, 2003.

[20] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD*, pages 805–810. ACM, 2005.

[21] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Managing Google's data lake: an overview of the Goods system. *IEEE Data Eng. Bull.*, 39(3):5–14, 2016.

[22] J. Heer, J. M. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.

[23] M. A. Hernández, P. Papotti, and W. C. Tan. Data exchange with data-metadata translations. *PVLDB*, 1(1):260–273, 2008.

[24] V. Kantere, D. Bousounis, and T. K. Sellis. A tool for mapping discovery over revealing schemas. In *EDBT*, 2009.

[25] N. Konstantinou, M. Koehler, E. Abel, C. Civili, B. Neumayr, E. Sallinger, A. A. A. Fernandes, G. Gottlob, J. A. Keane, L. Libkin, and N. W. Paton. The VADA architecture for cost-effective data wrangling. In *SIGMOD*, pages 1599–1602. ACM, 2017.

[26] J. Madhavan, P. A. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *ICDE*. IEEE, 2005.

[27] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In *VLDB*, pages 77–88, 2000.

[28] T. Milo, S. Novgorodov, and W. Tan. Interactive rule refinement for fraud detection. In *EDBT*, pages 265–276, 2018.

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

[30] B. ten Cate, P. G. Kolaitis, K. Qian, and W. Tan. Approximation algorithms for schema-mapping discovery from data examples. *ACM Trans. Database Syst.*, 42(2):12:1–12:41, 2017.

[31] I. G. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino. Data wrangling: The challenging yourney from the wild to the lake. In *CIDR*, 2015.

[32] R. Wisnesky, M. A. Hernández, and L. Popa. Mapping polymorphism. In *ICDT*, pages 196–208, 2010.