Meta-Mappings for Schema Mapping Reuse

Paolo Atzeni[§] Luigi Bellomarini[±] Paolo Papotti[∞] Riccardo Torlone[§]
[§] Università Roma Tre [±] Banca d'Italia [∞] EURECOM
{atzeni,torlone}@dia.uniroma3.it luigi.bellomarini@bancaditalia.it papotti@eurecom.fr

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 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 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 suitable to capture enterprise knowledge from previously defined schema mappings. We develop the techniques to infer meta-mappings from existing mappings, to organize them into a searchable repository, and to leverage the repository to propose to the 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(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/TBD

1. INTRODUCTION

Schema mappings are widely used as a principled and practical tool for data exchange and data integration. Although there are systems supporting data architects in the creation of mappings [12, 10], designing them is still a time-consuming task.

Given the overwhelming amount of "enterprise knowledge" stored in traditional data warehouses and in data-lakes [14, 18], reuse is an opportunity of increasing importance [1]. In particular, data management scenarios, such as data transformations, are often defined over schemas that are different in structure but share semantics. A great opportunity to reduce the effort in the design step is therefore to reuse existing schema mappings. Unfortunately, there is no obvious approach for schema mapping reuse. Consider the following example inspired from a real use case.

Proceedings of the VLDB Endowment, Vol. 12, No. xxx

DOI: https://doi.org/TBD

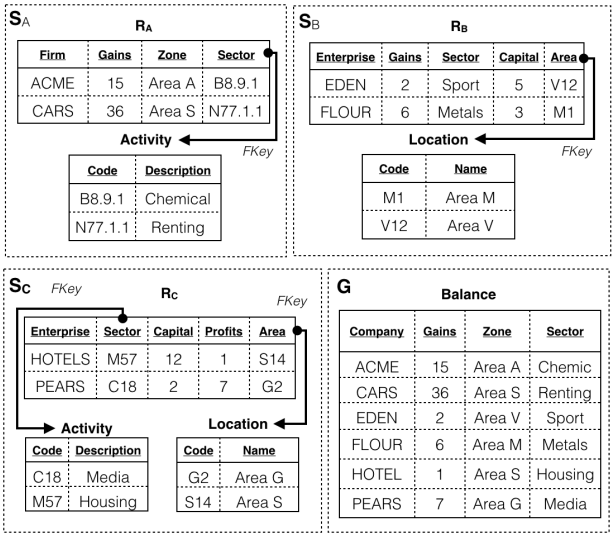


Figure 1: The business register at a central bank, G, and three data providers.

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 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, area of operation, and economic sector, whose code in* R_A *refers to relation* `Activity`*. 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`*. Data is moved from* $\mathbf{S}_A$ *and* $\mathbf{S}_B$ *into* $\mathbf{G}$*, by using two schema mappings:*

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

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

The example shows a *data exchange scenario* where the differences in the mappings are due to the structural differ-

ences 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 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 find and retrieve a suitable mapping.

Unfortunately, this form of direct reuse is challenging because of the nature of schema mappings. A mapping characterizes exchange and 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. For example, σ_A and σ_B cannot be directly re-used on $\mathbf{S}_C$. 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. Although there is little hope to re-use a schema mapping in its original form, there is a solution to extract a more general specification that is sound w.r.t. the original semantics.

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 fk 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. It seems a valid compromise between *precision*, i.e., a transformation that expresses the semantics of the original mapping, and *generality*, as it can be applicable over related 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 names g and a, and so they could be bound to any of `Gains`, `Sector` and `Capital`, incorrectly trying to map `Capital` into the target. The same problem applies with the generic mapping naively derived from σ_B.

EXAMPLE 3. *Consider 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 fk 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 that there is 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 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. However, three main challenges make reuse hard:

- 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 blowup 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.

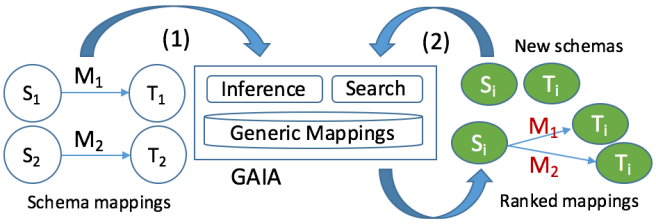


Figure 2: The architecture of GAIA.

In this work, we address the above challenges with GAIA, our system for *mapping reuse*. GAIA supports two tasks, as shown in Figure 2: (1) infer a generic mapping, *meta-mapping*, from one (or more) input schema mappings, and store it in a repository; (2) given a source and a target schema, return a ranking of suitable meta-mappings from the repository.

The system is based on 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 provide an algorithm to *infer* meta-mappings from schema mappings with an approach that extends previous efforts for the definition of schema mappings by example (Section 3).
- A retrieval method that, given a pair of schemas and a repository of meta-mappings, efficiently finds the most suitable meta-mapping and instantiates it. In particular, we introduce a feature-based metric, the *coverage*, that quantifies how suitable a meta-mapping is for a new scenario (Section 4). In fact, we use coverage to rank alternative meta-mappings as well as to index them to speed up the retrieval (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).

2. BACKGROUND AND GOAL

In this section we recall the notion of schema mapping [13] and introduce that of *meta-mapping*. While the former notion is used to model specific data transformations, the latter

introduces an abstraction over mappings [20, 26] 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, $M \subseteq Inst(\mathbf{S}) \times Inst(\mathbf{T})$ [8].

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. Given I and σ, if (I, J) satisfies σ then 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 [13]: 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 $h(J) = J'$.

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 mappings that will be discussed throughout the paper. A mapping between $\mathbf{S}_A$ and $\mathbf{G}$ is the st-tgd (where we omit quantifiers for the sake of readability):*

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

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 is a relation with the first two tuples in relation `Balance` *in Figure 1.*

A mapping from $\mathbf{S}_B$ to $\mathbf{G}$ is represented by 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 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 [26]. 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 in the running example (recalled in Example 4).*

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 (whichever they be) linked by a foreign key F and generates a target relation T obtained by joining R and S on F and taking 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. *Chasing m-schema $\mathsf{S_A}$ with Σ_A, both in Figure 3, generates the following target m-schema where $\perp_R$ is a labelled null denoting a placeholder for 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 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 *egd* [13] constraint 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, it is not only possible to generate a target schema by using the chase, as shown in Example 7, but that it is also possible to automatically obtain a schema mapping σ that represents the *specialization* of Σ for $\mathbf{S}$ and $\mathbf{T}$ [26]. The "Schema to Data Exchange Transformation" (S-D transformation) generates from $\mathbf{S}$ and Σ a complete schema mapping made of $\mathbf{S}$, a target schema $\mathbf{T}$ (result of chasing the source m-schema with the meta-mapping) and s-t tgd σ between $\mathbf{S}$ and $\mathbf{T}$. The variable correspondences in the tgd σ are derived from the provenance information computed during the chase step, in the same fashion of the provenance computed over the instance when chasing schema mappings [11].

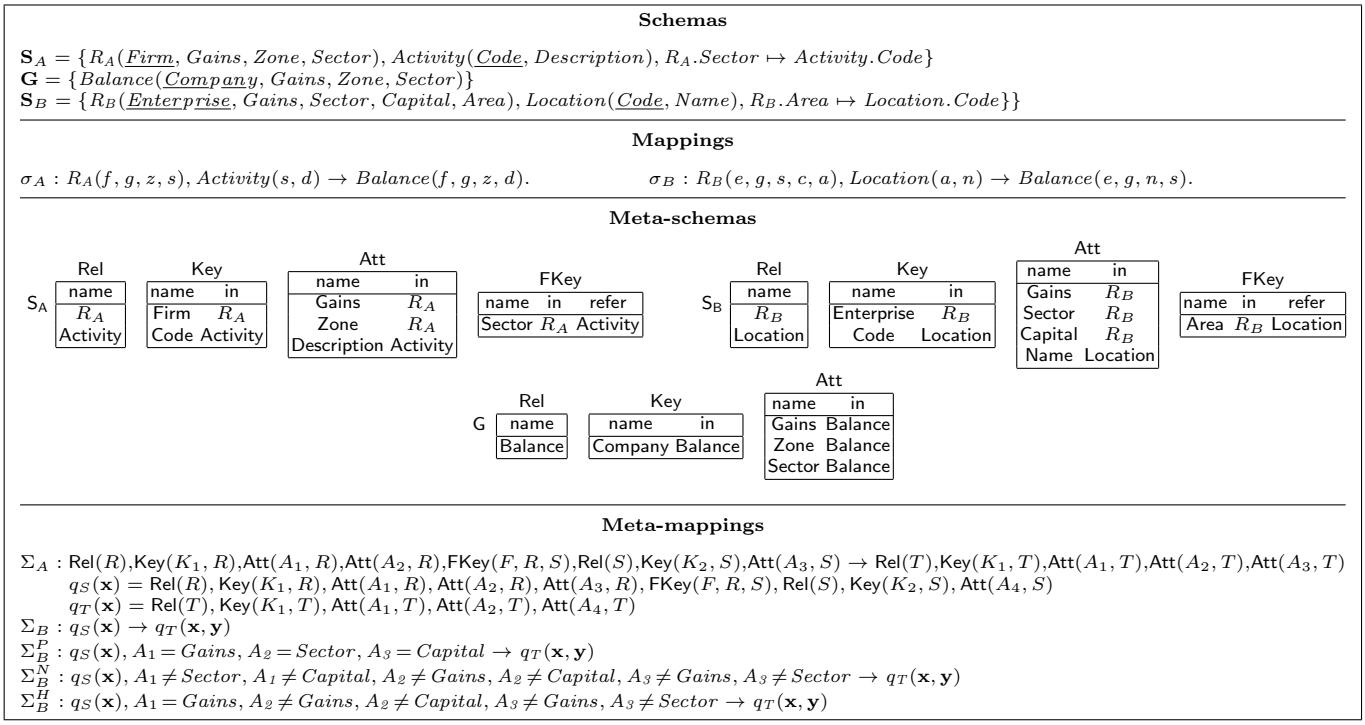


Figure 3: Schemas, m-schemas, mappings, and meta-mappings discussed along the paper.

EXAMPLE 8. *Consider again the scenario in Figure 3. If we apply the S-D transformation to schema* $\mathbf{S}_A$ *and 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, mapping σ_A *in Figure 3 from which meta-mapping* Σ_A *originates.*

2.4 Problem statement

While previous work focused on generating a schema mapping from a given meta-mapping [26], here we tackle the more general problem of *mapping reuse*, which consists of: (i) generating a repository of fitting 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, given $\mathbf{S}_A$ and $\mathbf{G}$, the goal is to reuse meta-mapping Σ_B, which has been inferred from σ_B.

3. FROM MAPPINGS TO META-MAPPINGS

In this section we describe how a schema mapping σ is generalized into a meta-mapping Σ. We start by introducing the notion of fitness, which formalizes the relationship between a meta-mapping and the mappings it generalizes. We then discuss how we infer fitting meta-mappings.

3.1 Fitness of meta-mappings

An important property of a meta-mapping is the ability to generate a mapping that is suitable for a given pair of source and target schemas, as formalized by property of *fitness*.

As a preliminary notion, we say that there is a *matching* between two m-schemas if there is an isomorphism between their tuples that preserves the identity on constants.

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* $\mathsf{S_A}$ *under* Σ_A *by chasing* $\mathsf{S_A}$ *using* Σ_A*. The result is the following:*

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

It is now easy to see that there is a isomorphism i *between* T *and* G *(in Figure 3), which maps labelled null* $\perp_B$ *to* Balance, Description *to* Sector*, and the other constants with the identity. It follows that, indeed,* Σ_A fits σ_A *according to* i*.*

For simplicity, we define the matching using the identity on costants. However, the definition can be expanded with a Boolean operator based on more relatexd notions of similarity, such as those used in schema matching [7]. Notice that our approach does not make any assumption on identity (or similarity) of attribute names, as shown in the example with the isomorphism connecting Description and Sector.

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 by equating, for each variable x that occurs in both the LHS and the RHS of σ, all the attributes in S and T on which x occurs, using the same distinct labelled constant.

EXAMPLE 10. *The projections of σ_A in Figure 3 on its source and target m-schemas are the following.*

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

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

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

Rel
name
Balance

Key	
name	in
c_f	Balance

Att	
name	in
Gains	Balance
Zone	Balance
c_d	Balance

For instance, constant c_f describes the fact that the σ_A maps the key of relation R_A to the key of Balance.

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 $\mathbf{T}$ 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 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 in T^σ 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 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 $\mathsf{S_B}$ (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
c_e	R_B
Code	Location

Att	
name	in
Gains	R_B
Sector	R_B
Capital	R_B
c_d	Location

FKey		
name	in	refer
Area	R_B	Location

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

Rel
name
Balance

Key	
name	in
c_e	Balance

Att	
name	in
Gains	Balance
Sector	Balance
c_d	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 is the following:

T

Rel
name
$\perp_B$

Key	
name	in
Firm	$\perp_B$

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

It is easy to see that this m-schema does not match with G.

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 and 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 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 Enforcing fitness

Let us now come to the problem of transforming a non-fitting canonical meta-mapping into a fitting meta-mapping.

To this aim, 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) $x_i = c_i$ or (ii) $x_i \neq c_i$, in which x_i is a variable and c_i is a constant.

We consider in particular mappings involving equality and inequality constraints, which we call *explicit mappings*. We denote them as *positive*, *negative* or *hybrid* 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 Example 8, 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.*

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

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

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

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 14 are likely to be more suitable in application domains involving `Gains` and `Capital` rather than those involving `Students` and `Courses`. Negative mappings cover more cases but tend to be too general, while positive mappings are more restrictive but better 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.

Note that, using the above constraints, there is the risk of generating 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 increases the chances to find relevant solutions.

In the next subsection we show how to turn a non-fitting canonical meta-mapping into a fitting meta-mapping by adding the constraints above. Unfortunately, this process unavoidably requires to detect all the ambiguities, a process with inherent exponential complexity.

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.*

3.4 Meta-mapping inference

We are now ready to present a technique to infer fitting meta-mappings from mappings: Algorithm 1 takes as input a transformation scenario $\mathscr{H}$ and returns a set meta-mappings that fit all the mappings in $\mathscr{H}$.

Algorithm 1 Meta-mapping inference.

```
1: procedure INFER
2: Input: a trasformation scenario H = (σ_1, ..., σ_n) where σ_i is a mapping between S_i (whose m-schema is S_i) and T_i, (whose m-schema is T_i)
3: Output: a set of meta-mappings Q fitting H
4: Q ← ∅
5:     for σ_i in H do
6:       (S_i^{σ_i}, T_i^{σ_i}) ← project-mapping(σ_i, S_i, T_i)
7:       Σ_i ← canonical-meta-mapping(S_i^{σ_i}, T_i^{σ_i})
8:       R_i ← repair(Σ_i, S_i^{σ_i}, T_i^{σ_i})
9:         for Σ in R_i do
10:            if Σ fits every σ in H then Q ← Q ∪ {Σ}
11: 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 6). We then build from $\mathsf{S}_i^{\sigma_i}$ and $\mathsf{T}_i^{\sigma_i}$ the canonical meta-mapping Σ_i (line 7), which may not fit σ_i. We then "repair" it by adding explicit mappings, as discussed in Section 3.3, and produce a set $\mathscr{R}_i$ of meta-mappings, all fitting σ_i (line 8), as described in detail by Algorithm 2 and discussed next.

As a final step of the main algorithm, we select, among the generated meta-mappings, those that fit not only the mapping from which they originate, but all the other mappings in the original scenario $\mathscr{H}$ (line 9 and 10).

Algorithm 2 describes how to repair a non-fitting canonical meta-mapping $\Sigma = q_S \rightarrow q_T$ by adding explicit mappings to Σ. First, 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 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 n-uples of potentially ambiguous variables are indeed ambiguous. The analysis is limited to extending homomorphisms since the others do not produce any 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 only the 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

Algorithm 2 Meta-mapping repair.

procedure EXPLICIT_REPAIR
Input: a meta-mapping Σ and a pair of m-schemas S and T
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 pot. ambiguos var. 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 pot. ambiguos var. in q_T
  for $h \in \mathscr{H}_E$ **do** ▷ here we detect ambiguous var.
    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 $(a, \cdot) \in \bigcap_{i=1}^n \tau(q_S, x_i)$ ▷ ambiguity in the LHS
        and $((x_1, \ldots, x_n) \notin \Lambda$
          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 h' extension of $h)$ ▷ w/o 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 $\mathscr{R}$

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 are *positive repairs*; cases with only inequalities are *negative repairs*; otherwise we have *hybrid repairs*.

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, 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 Figure 3.*

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). We now state the correctness of our procedures.

THEOREM 3. *Given a canonical meta-mapping Σ defined on m-schemas S^σ and T^σ, a (positive or negative) meta-mapping Σ_R that fits σ always exists and Algorithm 2 always terminates and determines such repair.*

4. MAPPING REUSE

We now introduce our solution to mapping reuse, given a new pair of schemas and a repository of meta-mappings. We start defining the approach (Section 4.1) and we then discuss how to implement it in an efficient and robust way (Section 4.2).

4.1 A general approach to reuse

Assume that we have inferred, from a set of mappings $\mathscr{H}$, a repository of meta-mappings $\mathscr{Q}$ using Algorithm 1 and we need to find a mapping for a new pair of source and target schemas, $\mathbf{S}$ and $\mathbf{T}$.

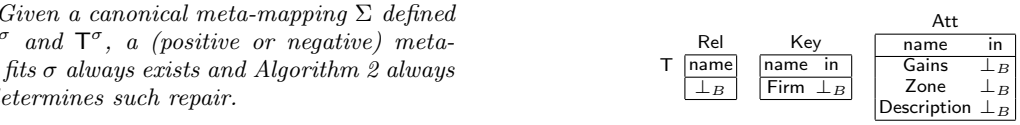
We can reuse the knowledge in the mappings $\mathscr{H}$ using $\mathscr{Q}$ as follows:. For each meta-mapping Σ in $\mathscr{Q}$

1. we generate, from $\mathbf{S}$ and Σ, a target schema $\mathbf{T}_\Sigma$ and a mapping σ from $\mathbf{S}$ to $\mathbf{T}_\Sigma$ using the S-D transformation, and

2. if Σ fits σ as a mapping between $\mathbf{S}$ and $\mathbf{T}$, we suggest σ as a possible solution.

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.*

The application of the S-D transformation *to $\mathsf{S_A}$ and Σ_B^H produce the following target m-schema T and corresponding mapping σ:*



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

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

Figure 4: Usability of meta-mappings.

To test whether Σ_B^H fits σ as a mapping between $\mathbf{S}_A$ and $\mathbf{G}$ we need to check the isomorfism between the m-schemas of T *and* G*, with the latter reported here for convenience:*

G

Rel
name
Balance

Key	
name	in
Company	Balance

Att

name	in
Gains	Balance
Zone	Balance
Sector	Balance

it is easy to see that there is an isomorphism between the tuples of G *and* T*. We then return σ between $\mathbf{S}_A$ and $\mathbf{G}$ to the user as a possible solution.*

Consider now Figure 4, where the columns denote meta-mappings and the rows pairs of source and target schemas. The table 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. Besides the pair of schemas and meta-mappings we have already introduced, we now discuss more 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, such as a modified version of G: $\mathbf{G} = \{Balance(\underline{Company}, Gains, Area, Sector)\}$, where one attribute label has been changed from "Zone" to "Area". Meta-mapping Σ_B^H can still be applied on $\mathsf{S_A}$ producing the following target m-schema $\mathsf{T} = \{Balance(\underline{Firm}, Gains, Zone, Description)\}$. Notice that with this m-schema, there are two possible isomorphisms with $\mathbf{G}$: one maps "Zone" to "Area" and "Description" to "Sector" (correct), and a second one maps "Zone" to "Sector" and "Description" to "Area" (incorrect). In these cases, we expose to the users both resulting mappings and let them decide.

No reusable meta-mappings. Consider now a new scenario one, 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 the attribute and the 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 result 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 that fully specify the attribute names, Σ_B^O, 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, 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 $\mathbf{G}$. In fact, it is clearly overfitting, in the sense that it is built from an exact correspondence to schema elements of σ_B and works only for this case.

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

4.2 Efficient top-k search

The approach above for the retrieval of reusable mappings is Boolean: a mapping either fits or not. However, it returns empty results when a meta-mapping needs only minor adjustments to fit the schemas. For instance, a meta-mapping that leads to only one isomorphism is preferable to another that is needs the user to verify multiple possible matchings. This behavior can be due to structural properties or to constants in the meta-mapping matching the source schema. It is clear that a ranking of the "best" meta-mappings for a specific scenario is indeed desirable, as it can guide the user selection.

Unfortunately, this is very expensive in terms of execution time with a large number of meta-mappings in the repository. The retrieval of a suitable mappings illustrated above requires to scan through all the stored meta-mappings and test both their fitness and their target schema isomorphisms for the given pair of schemas. While the scan of the entire repository is linear in the number of meta-mappings, both the fitness (based on the chase procedure) and the isomorphism tests have PTIME complexity.

To overcome the limitations above, we adopt a machine learning (ML) approach that, given a source and target schema pair, ranks as top results the meta-mappings that lead to fitting mappings for them. This solution not only enables efficient retrieval, but it also effectively ranks meta-mappings that are not fitting according to the amount of change needed to make them usable for the given input schemas. The approach is based on three phases:

- a *feature engineering phase*, in which 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 training phase*, in which a set of schemas and meta-mapping pairs labeled as fitting/non fitting are used to learn a distance metric (*coverage*) that scores the fitness of a meta-mapping for a pair of schemas;
- a *test phase*, where the distance metric is applied on an input pair of source and target schema and ranks the meta-mappings that best fit the given scenario.

As the fitness prediction for a pair of schemas and a meta-mapping takes constant time, the test phase takes linear time in the worst case, thus addressing problem (1). The distance metric enables top-k ranking of the most reusable meta-mappings, thus addressing problem (2). Furthermore, a distance metric allows the use of efficient data structures for fast retrieval of top ranked mappings. We exploit this property to design a solution based on a *k-d index* that operates in LOGTIME. We detail our solution in the next Section.

5. A REPOSITORY OF META-MAPPINGS

We start by describing the features that we identified to describe meta-mappings and schemas (Section 5.1). Then, we introduce the notion of *coverage*, a distance metric between meta-mappings specifically designed for the rank-by-reusability task (Section 5.2). Finally, we describe the optimizations to speed up the search over the repository (Section 5.3).

5.1 The features

We structure our features according to two categories. **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 (or number of relations in a schema with a homonym attribute). Some of them specifically refer to the LHS or the RHS of the meta-mapping, interpreted as the source and target schemas, respectively, when applied to schemas. In the Appendix, Figure **??** reports the list of structural features. **Domain features** represent the domain of interest of a specific meta-mapping or schema. We consider the set of all the constant names (of attributes, relations, etc.) used in the meta-mapping or in the schema.

5.2 The Coverage distance-metric

Although fitness is a solid indicator of the reusability of a meta-mapping w.r.t. a specific scenario, it is hardly applicable for repository search because of the polynomial complexity and the binary nature of its test.

We therefore introduce the notion of *coverage*, a distance-based metric that enables comparison *between meta-mappings* as well as *between meta-mappings and schemas*. The first comparison is used to organize and index the meta-mappings in the repository by contrasting their LHS and RHS. The second comparison is used to to set side by side the LHS and the RHS of a meta-mapping with a source and a target schema, respectively. 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 notions of structural and domain distance, for structural and domain features, respectively.

Structural distance measures the distance along a structural feature f_Σ of the LHS of a meta-mapping, and the corresponding one f_S for the source schema. It is defined as a $[0,1]$-normalized distance:

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

Domain distance measures the distance along a domain feature, i.e., it scores how likely the LHS of a meta-mapping and a source schema 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 a constant and the same constant is as an element in the schema. For example, given $A_1 = \texttt{Gains}$ and atom $\mathsf{Att}(A_1, R)$ in meta-mapping Σ, there is 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. More specifically, we adopt the *Jaccard distance* of the sets of constants:

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

We can now define our *coverage metric* χ.

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

For every pair of features to compare, we interpret 1-distance (namely, the *score*) as the probability that two 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 for a pairs of features we have a 0.5 score, then such feature is not informative for the coverage, i.e., it does not add anything to the a random baseline. 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 actually built as the *Graham combination* [17] of the scores for all features.

Classifier Training. The supervised training to learn our metric consists in the application to every score of a parametric scaling formula, which adjusts the positive or negative contribution of the score by altering the definition range.

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, we tune a and b for each feature 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.

In our model, parameters can be learned in many ways with typical ML techniques. In particular, we train a *logistic classifier* [9] with a training set consisting of pairs $\langle \mathbf{X}, Y \rangle$, where $\mathbf{X}$ is the vector of scores for each sample and Y the target variable denoting whether we are in a fitting situation (hence the value one), or in a non-fitting situation (value zero). For each sample, the following formula applies:

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

where $\text{logit}(p) = y = \ln\frac{p}{1-p}$, with p being the fitness probability and β the vector of b parameters in the feature scaling. Solving with respect to β, for each feature we obtain the logarithmic odds, i.e., the marginal relevance of a specific feature f, from which we can derive $b = \frac{e^{\beta f}}{1+e^{\beta f}}$. The same is done to calculate a. Values for a and b for each feature are the values for the parameters in χ. We experimentally show that effective configurations can be identified with very limited training data.

Notice that the estimated parameters could be directly used in the so-trained logistic classifier for fitness estimation. However, we adopt the distance metric for ranking, as the input schemas can be compared with the meta-mappings in the repository and the system is able to return the top-k meta-mappings by directly using the coverage. Moreover, coverage enables the optimization techniques to efficiently retrieve from the repository the most covering meta-mappings, as we discuss next.

5.3 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

Table 1: Transformations and schemas statistics.

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

as a point of a multi-dimensional space, which we index by means of a *k-d tree* [6], 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 5.2, with only structural features.

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 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.

6. EXPERIMENTAL EVALUATION

We evaluate our system, GAIA, on real-world 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. In Section 6.3, we show the effectiveness of the coverage in ranking results. Finally, in Sections 6.4 and 6.5, we report on the quality of the returned transformations in large scale experiments with real mapping scenarios.

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.

Datasets and Transformations. 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

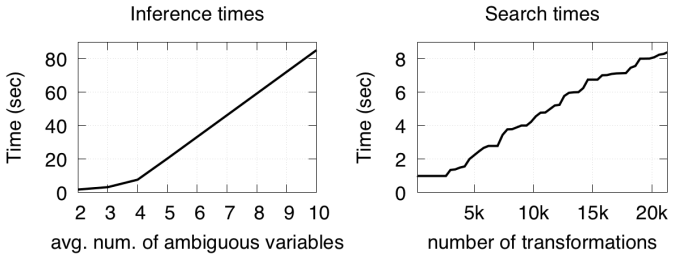


Figure 5: Inference and search execution times.

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.

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 5(a), we report the times for computing the inference of meta-mappings. We run the test inferring the meta-mappings from the original mappings 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 meta-mappings are stored in the repository. For a large number of variables, it take about a minute for 10 ambiguous variables.

We then evaluate the search time for the three use cases on the same repository, as reported in Figure 5(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.

6.3 Coverage and fitness

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

Consider again the 4 mappings $(\sigma_A, \sigma_B, \sigma_C, \sigma_D)$ and a repository with the 5 meta-mappings $(\Sigma_A, \Sigma_B, \Sigma_B^P, \Sigma_B^N, \Sigma_B^H)$ described in the previous examples (Figure 4). For each pair of source and target schemas (σ_x), we retrieve from the repository the list of meta-mappings $(\Sigma_Y, \ldots, \Sigma_Z)$ ranked according to their coverage for the given scenario. We compute the feature scores for this experiment by using the 20 examples in the training set (we discuss later the impact of the number of training examples on feature scores).

For every pair of (source, target) schemas in the original mapping scenario, Table 2 reports the number of fitting meta-mapping in the repository and the precision in the top-k results ranked according to their coverage (with k= 1, 2,

Table 2: Precision@k according to coverage.

Scenario	# of fitting m-m	k=1	k=2	k=3	k=4
σ_A	4/5	1.0	0.75	0.75	0.75
σ_B	3/5	1.0	1.0	1.0	0.75
σ_C	2/5	1.0	1.0	0.67	0.5
σ_D	0/5	0	0	0	0

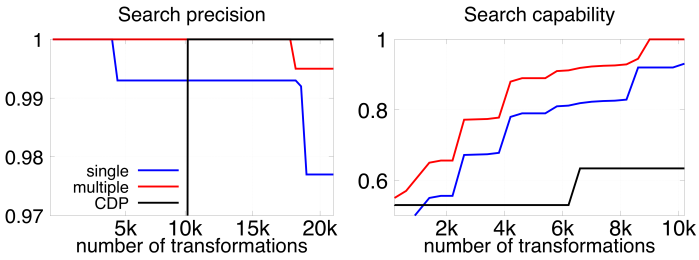


Figure 6: Search precision and capability.

3, 4). The results show that the coverage rank at the top the fitting meta-mappings, when they are in the repository for the given scenario. In particular, the results are perfect for σ_B and σ_C. For σ_A, for $k = 1$ a fitting meta-mapping has the highest score, but three meta-mappings follow it in the ranking with the same score. Since among these three there is one that is not fitting, the accuracy is then 0.75 for k=2, 3, 4. Indeed, it is not possible to distinguish Σ_B^P, Σ_B^N and Σ_B^H with our features only. The non fitting meta-mapping can be distinguished with expensive homomorphism checks, but we believe the trade-off offered by our features in terms of accuracy is overall positive given their very low execution time. Finally, for σ_D there are no fitting meta-mappings in the corpus. This explain why the precision is always 0, this reflects the quality of the corpus w.r.t. σ_D and not the quality of the coverage as ranking mechanism. However, it is important to notice that the coverage also gives a normalized measure of fitness. This is crucial for two reasons. First, it is effective in identifying meta-mappings that are close to being fit for a given scenario. While Σ_B^P is not fit for σ_A, its coverage is 0.49, as a single change in a constant would make it fit. All the meta-mappings in the repository are not fitting for σ_D and their coverage is always lower than 0.39. Second, the score enables a different application from ranking, where we make binary decision on the fitness based on a threshold. In the experiment, by manually setting a threshold value (t=0.48) to classify fitting and not fitting meta-mappings, the coverage correctly labels all the meta-mappings except one for a precision of 0.95 in the classification. We use this threshold value to make binary decision in a more specific experiment to evaluate the impact of training over the features in our model (see Appendix **??**).

The experiments with labeled scenarios show that coverage is an effective measure to rank meta-mappings. As labeling fitness for meta-mappings 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

For each mapping σ on a source S and a target T, the repository stores a set of new meta-mappings $\mathcal{Q}$, inferred from σ. We measure with the *precision* the ability of the system to return the meta-mappings in $\mathcal{Q}$ when queried with S, T, i.e., it measures how well the system retrieves correct 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 $\mathcal{Q}_\sigma$. We then pick a schema mapping σ' from Θ and search the repository by providing as input the source and target schemas in σ'. As a query returns thousands of applicable meta-mapping, we rank them according to the coverage and take the top 10 (k=10). We collect the top-10 result and compute the percentage of rightly retrieved meta-mappings Σ, i.e., $\Sigma \in \mathcal{Q}_{\sigma'}$.

We test search precision with a repository of increasing size (from 200 to 21,301). In each test, we query the repository by using 50 different source-target pairs and report their average precision. 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). 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.

In Figure 7(c), precision decreases with the size of the repository as larger numbers of mappings lead to an increasing number of false positives in the search result. We report the precision in retrieving the correct mapping for CDP with a separate line. As soon as the 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 specific structures and constants lead to meta-mappings that are easy to retrieve.

Inferring meta-mappings from multiple schema mappings (multiple scenario) improves precision 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.

6.5 Search capability

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 capability*. Given a mapping σ from S to T and the set of meta-mappings $\mathcal{Q}_\sigma$ inferred by σ, we consider S, T and search the repository, making sure that it does not include any meta-mapping of $\mathcal{Q}_\sigma$. We then measure the search capability as the coverage (as defined in Section 5.2) of the k best retrieved meta-mappings w.r.t. the meta-mappings in $\mathcal{Q}_\sigma$.

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 as input. We then compute the average α of the coverage of the top-10 meta-mappings (k=10) w.r.t. the meta-mappings $\mathcal{Q}_\sigma$. 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 repository and the transformation scenario. 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

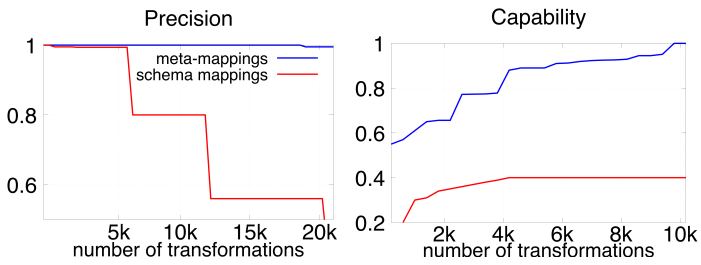


Figure 7: Precision and capability for meta-mappings and schema mappings.

the system provides transformations for it. We then compute the search capability with 50 different source-target pairs. In this case, we explicitly run the experiment using CDP.

As apparent in Figure 7(d), the more mappings are added to the repository, the higher is the search capability. 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 after the inclusion of the transformation number "6400", 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 capability for all configurations. This shows the positive impact of the merge procedure.

These tests show that 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.

6.6 Schema mappings corpus

Finally, we repeat the experiments on the real-world dataset and use case **S-T** to measure search precision 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 (e.g., Σ_B^O in Section 4). The results for the meta-mappings are the ones we obtained in the previous experiments with multiple configurations.

In Figure 7(g), the precision results with a schema mapping repository are comparable to meta-mappings up to about 13000 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 capability (Figure 7(h)). For new, unseen cases, a repository of schema mappings does not provide significant benefit, never going above 0.4 for any configuration. This demonstrates the specificity of schema mappings, that are hardly applicable for semantically similar cases.

7. RELATED WORK

While the notion of reuse is popular for software components [15], it is getting 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* [14, 18] or *knowledge graphs* [5], and from the wish to activate "enterprise knowledge" by means of complex data management and reasoning tasks. A recent example of the reuse approach is the automatic adaptation of rules for fraud detection across different datasets [25].

Our work goes towards this direction by introducing the first framework for the reuse of schema mappings. In particular, we adopt the standard language of tgds for the declarative definition of data transformations in data exchange [13], enriched with the specific semantics of meta-mappings for the problem of exchanging metadata instead of data [26]. One contribution in data exchange is the semi-automatic generation of the schema mappings [12], given simple correspondences between the schema elements [7, 4]. Matching discovery and generation support data transformation design [21], but they must be done for each new pair of schemas. Moreover, 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 [23], but they cannot store logical formulas and therefore crucial information, such as the structure of the schema and the manual tuning, is completely lost, as experimetally demonstrated in Section 6.6.

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, 16, 27], which in turn is inspired by the discovery of queries given data examples [24]. In our setting, previous algorithms [2, 3] would generate canonical meta-mappings that, as discussed in Section 3, would not capture the semantics of a transformation. Our repair algorithms (Section 3) can be seen as a generalization of the previous algorithms where, with a limited extension of the adopted language, we also deal with data examples with self-joins. Gottlob and Senellart [16] introduce a theoretical framework for the discovery of schema mappings, based on a single data example. They address unsuitable mappings by examining different repairs and solving a cost-based optimization problem to choose the best ones. Their results are not directly applicable to our context for two main reasons: first, 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. Kolaitis et al. [27] build on Gottlob's framework and propose polynomial-time approximation algorithms to quickly choose the best repairs. This is certainly useful in practice, yet would be limiting in our setting, where we are actually interested in storing all the variants to maximise reuse.

Traditional ETL tools and research and commercial systems for *data wrangling* such as VADA [22] and Trifacta [19] 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 transformations.

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. Experiments confirm the high quality of the retrieved mappings.

One direction for future work is the extension of the approach to model constants and function symbols in the schema mappings. 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] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer, 2011.

[5] 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.

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

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

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

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

[10] 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.

[11] L. Chiticariu and W. C. Tan. Debugging schema mappings with routes. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 79–90, 2006.

[12] 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.

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

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

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

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

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

[18] 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.

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

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

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

[22] 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 Conference*, pages 1599–1602. ACM, 2017.

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

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

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

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

[27] 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.