A Collective, Probabilistic Approach to Schema Mapping

Angelika Kimmig
KU Leuven
angelika.kimmig@cs.kuleuven.be

Alex Memory
University of Maryland
memory@cs.umd.edu

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

Lise Getoor
UC Santa Cruz
getoor@ucsc.edu

***Abstract*—We propose a probabilistic approach to the problem of schema mapping. Our approach is declarative, scalable, and extensible. It builds upon recent results in both schema mapping and probabilistic reasoning and contributes novel techniques in both fields. We introduce the problem of *mapping selection*, that is, choosing the best mapping from a space of potential mappings, given both metadata constraints and a data example. As selection has to reason holistically about the inputs and the dependencies between the chosen mappings, we define a new schema mapping optimization problem which captures interactions between mappings. We then introduce *Collective Mapping Discovery* (CMD), our solution to this problem using state-of-the-art probabilistic reasoning techniques, which allows for inconsistencies and incompleteness. Using hundreds of realistic integration scenarios, we demonstrate that the accuracy of CMD is more than 33% above that of metadata-only approaches already for small data examples, and that CMD routinely finds perfect mappings even if a quarter of the data is inconsistent.**

I. Introduction

Schema mappings are collections of complex logical statements which relate multiple relations across data sources with different schemas, and thus can be used to exchange data between these sources. Efficient techniques for reasoning about the suitability of different schema mappings are crucial to manage the massive number, complexity, and size of data sources. While the metadata and data of the sources often provide evidence for how to best map them, this evidence is rarely complete or unambiguous. To reason effectively about mappings, we thus need techniques grounded in mapping understanding that can reason about open-world scenarios using uncertain, imperfect evidence.

We study the problem of *mapping selection*, that is, of selecting from a large set of possible mappings, a mapping that best relates a source and a target schema. We define the mapping selection problem for the entire language of st tgds (source-to-target tuple-generating-dependencies; also known as GLAV mappings) which is arguably the most commonly used mapping language [1]. We then provide an efficient solution to this problem based on state-of-the-art probabilistic reasoning.

Historically, approaches to schema mapping discovery and selection have considered a wide variety of inputs. Early approaches use metadata (schema constraints) and attribute correspondences (aka schema matchings) to create mappings that are consistent with the metadata [2], [3]. Metadata in the form of query logs has been used to select mappings that are most consistent with frequently asked queries [4]. Many different approaches use data to refine a mapping or to select a mapping from among a set of schema mappings [5], [6], [7], [8], [9], [10], [11], [12]. Other approaches solicit user feedback to define scores for each view in a set of candidate views and then select an optimal set of views based on these scores [13]. All of these approaches have merit, but are tailored to a specific form of input evidence, and either work for limited mapping languages, like views, or assume consistent or complete input, which is difficult to prepare or find. An exception to this is the approach by Alexe et al. [12] that considers *bad* data examples that are consistent with several (candidate) mappings or none. They consider how such *bad* examples can be turned into *good* examples that are consistent with a single, desired mapping.

We define a new mapping selection problem that uses both data and metadata collectively as input. None of the evidence is required to be consistent or complete, rather we find the subset of st tgds that are best supported by the given evidence as a whole. Metadata can serve as a guide through a potentially massive set of possible mappings, suggesting mappings that are consistent with schema semantics (e.g., joining relations on a foreign key). Data can reinforce metadata evidence. Data can also rule out a mapping that is consistent with the metadata, but inconsistent with large parts of the data. Metadata can obviate the need to have two pristine data instances as input that precisely define a single *best* mapping. Furthermore, our framework is declarative and extensible to new forms of evidence including scores (such as user-feedback annotations) on the metadata and data evidence.

Our solution adopts and extends some of the latest techniques from the probabilistic reasoning community. These techniques are routinely used to combine logical constraints in relational domains with the ability to handle uncertainty and conflicting information. Building upon work of Gottlob and Senellart [18], we refine their concepts of validity and fully explaining to define what it means for a single tuple to be either an (incomplete) error for a mapping or (partially) explained by a mapping. Using these notions, we define our probabilistic optimization problem using probabilistic soft logic (PSL) [14], a scalable probabilistic programming language based on weighted logical rules. PSL has been used successfully for a variety of data and knowledge integration problems, including knowledge graph identification [15] and data fusion [16], [17], but did not support the kind of open world reasoning required for mapping selection. We therefore extend PSL with *prioritized disjunctions*, which provide a tractable framework for handling reasoning over tuples that may be explained by a mapping and thereby allow us to define key features of the mapping selection problem.

We refer to our solution as *Collective Mapping Discovery* (**CMD**), because it reasons collectively both about multiple forms of evidence and over the interactions between different st tgds. **CMD** advances the state-of-the-art in schema mapping in several important ways including integrating evidence at a much finer-grained level of detail than attempted in the past. In addition, the declarative nature of **CMD** makes it easy to extend in a variety of ways.

We perform an extensive empirical validation of our approach. We use the integration benchmark iBench [18] to test **CMD** on a wide variety and large number of mapping scenarios. We use IQ-METER [19], a multi-criterion evaluation measure, to confirm the quality of **CMD**'s output. We compare **CMD** with a baseline approach which uses only metadata. We show that the accuracy of **CMD** is more than 33% above that of a metadata-only approach already for small data examples. In addition, we illustrate the robustness of our approach by demonstrating that we are able to find accurate mappings even if a quarter of the data is dirty.

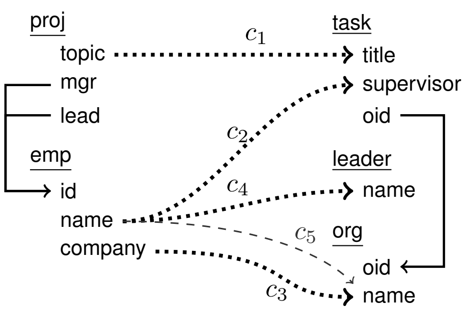
Section II illustrates the key challenges with an example. Section III introduces the selection problem for st tgds without existentially quantified variables, and Section IV extends this to st tgds. Section V introduces our solution using PSL and our extension of PSL with *prioritized disjunctions*. We discuss experiments in Section VI and related work in Section VII.

II. MOTIVATING EXAMPLE

Figure 1(a) shows a pair of source and target schemas, foreign keys (solid lines) and attribute correspondences (or matches, dotted lines), which we will use as a running example. The metadata is ambiguous, as it is not clear from the schemas whether task.supervisor in the target schema is associated with proj.mgr or proj.lead in the source schema. A data example in the form of an instance of the source schema (I) and an instance of the target schema (J) can help resolve such ambiguity. The data example in Figure 1(b), where org and leader are empty, suggests that supervisors in task tuples correspond to mgr in the source, not lead. Interactive schema mapping refinement techniques use data to select among a set of mappings. They take as input a set of candidate mappings and use data to interactively guide a user in selecting a subset that is correct [20], [5], or in correcting a set of data examples so that a "best fitting" mapping exists [21]. The interactive nature of these solutions permits a user to decide what mapping is best given metadata and data evidence. In contrast, we do this reasoning automatically to find the best fitting mapping.

We consider the problem of combining metadata evidence (in the form of a set of candidate mappings) and potentially imperfect data evidence (in the form of a data example) to select an optimal mapping. More specifically, our candidate mappings are source-to-target tuple-generating-dependencies (st tgds).[1] These are simple first-order logic statements relating a source query and a target query. The candidates may come from a mapping design tool like Clio [22] or ++Spicy [23], or may have been mined from a query log [4].

[1] The term *mapping* is often used both for a single st tgd and for a set of st tgds. Here, we use *candidate mapping* or *candidate* to refer to a single st tgd; while *mapping* generally refers to a set of st tgds.



(a) Source (left) and target schema (right) with corresponding attributes (dotted lines), a spurious correspondence (dashed), and foreign key constraints (solid lines).

proj

topic	mgr	lead
BigData	1	2
ML	1	1

task

title	supervisor	oid
BigData	Alice	111
ML	Alice	111

emp

id	name	company
1	Alice	SAP
2	Bob	IBM
3	Pat	MS

(b) Initial data example.

leader

name
Alice
Bob

org

oid	name
111	SAP
222	MS

(c) Additional data.

$$\theta_0 : \mathsf{proj}(\mathbf{t}, \mathrm{m}, \mathrm{l}) \land \mathsf{emp}(\mathrm{m}, \mathbf{n}, \mathrm{c}) \rightarrow \exists\, \mathrm{o}.\, \mathsf{task}(\mathbf{t}, \mathbf{n}, \mathrm{o})$$
$$\theta_1 : \mathsf{proj}(\mathbf{t}, \mathrm{m}, \mathrm{l}) \land \mathsf{emp}(\mathrm{l}, \mathbf{n}, \mathrm{c}) \rightarrow \exists\, \mathrm{o}.\, \mathsf{task}(\mathbf{t}, \mathbf{n}, \mathrm{o})$$
$$\theta_2 : \mathsf{proj}(\mathbf{t}, \mathrm{m}, \mathrm{l}) \land \mathsf{emp}(\mathrm{m}, \mathbf{n}, \mathbf{c}) \rightarrow \exists\, \mathrm{o}.\, \mathsf{task}(\mathbf{t}, \mathbf{n}, \mathrm{o}) \land \mathsf{org}(\mathrm{o}, \mathbf{c})$$
$$\theta_3 : \mathsf{proj}(\mathbf{t}, \mathrm{m}, \mathrm{l}) \land \mathsf{emp}(\mathrm{l}, \mathbf{n}, \mathbf{c}) \rightarrow \exists\, \mathrm{o}.\, \mathsf{task}(\mathbf{t}, \mathbf{n}, \mathrm{o}) \land \mathsf{org}(\mathrm{o}, \mathbf{c})$$
$$\theta_4 : \mathsf{emp}(\mathrm{i}, \mathrm{n}, \mathbf{c}) \rightarrow \exists\, \mathrm{o}.\, \mathsf{org}(\mathrm{o}, \mathbf{c})$$
$$\theta_5 : \mathsf{emp}(\mathrm{i}, \mathbf{n}, \mathrm{c}) \rightarrow \mathsf{leader}(\mathbf{n})$$
$$\theta_6 : \mathsf{proj}(\mathrm{t}, \mathrm{m}, \mathrm{l}) \land \mathsf{emp}(\mathrm{l}, \mathbf{n}, \mathrm{c}) \rightarrow \mathsf{leader}(\mathbf{n})$$
$$\theta_7 : \mathsf{proj}(\mathbf{t}, \mathrm{m}, \mathrm{l}) \land \mathsf{emp}(\mathrm{m}, \mathbf{n}, \mathrm{c}) \rightarrow \exists\, \mathrm{o}.\, \mathsf{task}(\mathbf{t}, \mathbf{n}, \mathrm{o}) \land \mathsf{org}(\mathrm{o}, \mathbf{n})$$
$$\theta_8 : \mathsf{proj}(\mathbf{t}, \mathrm{m}, \mathrm{l}) \land \mathsf{emp}(\mathrm{l}, \mathbf{n}, \mathrm{c}) \rightarrow \exists\, \mathrm{o}.\, \mathsf{task}(\mathbf{t}, \mathbf{n}, \mathrm{o}) \land \mathsf{org}(\mathrm{o}, \mathbf{n})$$

(d) Candidate st tgds. Variables in bold denote exchanged attributes.

Fig. 1: Motivating example; see Section II for details.

A key challenge in mapping selection is that the number of possible selections is exponential in the number of candidate st tgds. Consider the candidates in Figure 1(d), focusing first on our earlier data example (Figure 1(b)) and candidates θ_0 and θ_1. Notice that the data example is *valid* for θ_0 (meaning (I,J) satisfy the mapping θ_0) but is not *valid* with respect to θ_1, as there is no tuple (BigData, Bob, -) (with some oid). We call such a missing tuple an *error*. Errors might be caused by dirty data. The data example contains a tuple (BigData, Alice, 111) and this tuple may be dirty (the value of Alice is wrong should be Bob) causing this *error*. If the data is clean, this error tuple would suggest that we should prefer θ_0 over θ_1.

Note that θ_0 and θ_1 both ignore the correspondence between emp.company and org.name. Mapping θ_2 also explains the data (intuitively), but it explains more, as it creates org tuples for which we have no data evidence. If we change our data example to include the org tuples in Figure 1(c), the data suggests that we should select both θ_2 and θ_4. The mapping θ_2 alone maps the inner join of the source data to the target. Mappings θ_2 and θ_4 together map the right outer-join.

If we also add the leader tuples in Figure 1(c) to our data example, θ_5 explains all leader tuples. However, θ_5 is not *valid* with respect to the data, as it also suggests that tuple (Pat) should appear in leader, but it does not and thus is an error for θ_5. The mapping θ_6 addresses this by joining emp with proj via proj.lead; it both explains and is valid with respect to the leader example data. Generally, we seek sets of st tgds that collectively explain the data and are valid with respect to the data. On that basis, the set $\{\theta_2, \theta_4, \theta_6\}$ is a good choice.

Note that our candidates θ_0 - θ_6 use only correspondences c_1-c_4 in Figure 1(a). If a matcher has incorrectly suggested the correspondence c_5 : emp.name $\rightarrow$ org.name, then we may get additional candidate mappings like θ_7 or θ_8 that use this correspondence. However, in this example (and in many real examples) a small data example can eliminate such candidates, as they are likely not to explain the data or be valid.

This example illustrates many challenges in schema mapping discovery from metadata and data.

Dirty or Ambiguous Metadata. Our goal is to find a mapping that fits the metadata. In practice, the number of such mappings can be huge, due to metadata ambiguities such as 1) multiple foreign key paths between relations; 2) the choice between inner and outer joins; 3) the presence of bad correspondences. Dirty metadata (for example, incorrect foreign keys) exacerbates this problem. Data can help in selecting correct mappings. We tackle the problem of combining metadata and data evidence to effectively and efficiently select a mapping, even if the data does not fully disambiguate all metadata. In our example, we may have some target tuples that are consistent with a join on mgr (θ_0) and some that are consistent with a join on lead (θ_1); e.g., (ML, Alice, 111) is consistent with both θ_0 and θ_1. Our solution will weigh the evidence to find a mapping that is most consistent with the evidence as a whole.

Unexplained Data. We are given example source (I) and target (J) data and our goal is to find a mapping that explains the target data. In practice, we rarely have *perfect* data examples that only contain target data explained by I. Indeed, the open nature of st tgds permits the target to have independent data that was not derived from the source. For example, suppose there is a target org (333, BM), and the value BM does not appear in the source. This data may be correct data (the target has data about the Bank of Montreal and the source does not) or it may be dirty data (perhaps the value BM was mistyped and should be IBM). Even if no candidate explains these tuples, we still want to find the best mapping. So our optimization should not break (or fail) in the presence of some unexplained data. Furthermore, if there is a mapping that explains all data, we may not choose it if it is not valid with respect to the data example, or if it is considerably more complex (larger) than one that fails to explain a few tuples in J.

Data Errors. Our goal is to find a mapping that is valid for the given data example (I, J). Again, in practice, it is unrealistic to assume a data example that is *perfect* in this way. Hence, we provide a solution that is tolerant of some errors (for example, some dirty source data or some missing target data), but seeks to find a set of st tgds for which the errors are minimized (so it is as valid as possible with respect to the data).

Unknown Values. Our goal is to find a mapping that may use existential quantification where appropriate. This is challenging, as such mappings introduce unknown or null values in the target. For instance, st tgds θ_0 and θ_1 both only cover part of target tuple (ML, Alice, 111), as they cannot "guess" the value of the oid. Still, we need to compare them to st tgds that may have no existentials and therefore cover entire target tuples. This problem is made more challenging as existentials play a critical role in identifier (or value) invention where the same existential value is used in multiple tuples to connect data together. It is important that mappings that correctly connect the data be considered better than mappings that use different existentials. For example, we prefer θ_2 over the combination of θ_0 and θ_4, since the data supports the connection θ_2 makes between task and org in the target. This is an important aspect of the problem that has not been considered by earlier work on view (also known as full st tgd) selection [13].

To address these challenges, we present a fine-grained, scalable solution that gives an st tgd *credit* for each tuple it can explain or partially explain (in the case of existential mappings) and aggregates this information to find a set of st tgds that best explain the data. A set of st tgds is penalized for each error tuple (the more errors the less valid the mapping). Hence, we find the set of candidate st tgds that collectively minimize the number of errors and number of unexplained tuples, even under contradictory or incomplete evidence.

III. Selection over Full Mappings

We first define mapping selection for full st tgds [1], that is, st tgds without existentially quantified variables, and extend our definitions to arbitrary st tgds in Section IV.

A. Mapping Selection Inputs

We define our mapping selection problem with respect to a *source* schema $\mathbf{S}$ and a *target* schema $\mathbf{T}$, where a schema is a set of relations. The *data evidence* consists of a data example, that is, a pair of instances I of $\mathbf{S}$ and J of $\mathbf{T}$. The *metadata evidence* consists of a (finite) set $\mathcal{C}$ of candidate st tgds. An st tgd is a logical formula $\forall \mathbf{x}\ \phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$, where ϕ is a conjunction of atoms over the relations of $\mathbf{S}$ and ψ over those of $\mathbf{T}$ [1]. Here, $\mathbf{x}$ and $\mathbf{y}$ are sets of logical variables. If $\mathbf{y}$ is empty (no existentials) then the st tgd is a *full st tgd* [24].

Candidate st tgds can be generated using existing schema mapping systems. Such systems, both industrial systems (like Altova Mapforce or Stylus Studio) and research systems, generate sets of candidate mappings and generally let users select or refine these mappings using a variety of visual interfaces. To generate candidate mappings, research systems like Clio [22], HepTox [25], and ++Spicy [23] use schema constraints, while U-Map [4] uses query logs. By building on these existing approaches, we focus on candidate mappings that are plausible according to the metadata and the methodology used in candidate generation rather than all possible mappings.

B. Characterizing the Input Quality

Given metadata evidence ($\mathbf{S}$, $\mathbf{T}$, $\mathcal{C}$), our goal is to find a subset $\mathcal{M} \subseteq \mathcal{C}$ that best "fits" the data example (I, J). Let $\mathbf{C}$ be the set of all constants in $I \cup J$, and $\mathbf{N}$ a set of labeled nulls (disjoint from $\mathbf{C}$). Following Fagin et al. [24], a homomorphism between instances $h : K_1 \rightarrow K_2$ is a mapping from $\mathbf{C} \cup \mathbf{N}$ to $\mathbf{C} \cup \mathbf{N}$ such that: (1) for every $c \in \mathbf{C}$, $h(c) = c$, and (2) for every $R(t)$ of K_1, $R(h(t))$ is in K_2. A homomorphism $h : \phi(x) \rightarrow K$ is a mapping from the variables x to $\mathbf{C} \cup \mathbf{N}$ such that for every $R(x_1, \dots, x_n)$ in $\phi(x)$, $R(h(x_1), \dots, h(x_n))$ is in K. Let $\mathcal{M}$ be a set of st tgds, then an instance K of $\mathbf{T}$ is called a *solution* for I, if $(I, K) \models \mathcal{M}$. An instance K is a *universal solution* if it is a solution and if for every other solution K', there is a homomorphism $h : K \rightarrow K'$. Fagin et al. [24] showed how a universal solution can be computed efficiently using the chase over $\mathcal{M}$ (and such a universal solution is typically called a *canonical universal solution*).

Gottlob and Senellart [7] call a mapping $\mathcal{M}$ *valid* for (I, J) if J is a solution for I under $\mathcal{M}$. Suppose $(I, J) \not\models \mathcal{M}$. Intuitively, this means J misses tuples that must be in every solution for I. We call such tuples *errors*. A ground tuple t (that is, a tuple containing only constants) is a *full error* if it is not in J but in every J' such that $(I, J \cup J') \models \mathcal{M}$. If K is a universal solution for $\mathcal{M}$ and I, then t is a full error iff $t \in K$ and $t \notin J$. If (I, J) is valid with respect to $\mathcal{M}$ then there are no full errors.

Example 1: *The candidate θ_5 in Figure 1(d) is not valid with respect to the data example in Figure 1(c). However, if we add the tuple t' = leader(Pat)) to J then θ_5 is valid for $(I, J \cup t')$. Thus, (Pat) is a* full error*, and the only full error.* □

Ideally, all tuples in J should be explained, that is, be a result of the selected candidate mappings applied to I. Again following Gottlob and Senellart [7], a mapping $\mathcal{M} \subseteq \mathcal{C}$ and source instance I *explain* a ground fact t, if $t \in K$ for every K such that $(I, K) \models \mathcal{M}$. A mapping $\mathcal{M}$ and I *fully explain* J if they explain every tuple in J. A ground tuple t is explained by $\mathcal{M}$ and I iff t is in a universal solution for $\mathcal{M}$ and I. As with validity, we would like to permit exceptions, that is, a few tuples in J that *unexplained*, meaning not fully explained.

Example 2: *Consider again θ_5 of Figure 1(d). For the instance I of* emp *shown in (b), θ_5 fully explains J (the two* leader *tuples) shown in (c). However, if* leader *also contained* leader(Joe)*, then θ_5 would still be valid, but* leader(Joe) *is an unexplained tuple.* □

C. Collective Selection over Full Mappings

We now define an optimization problem for finding $\mathcal{M} \subseteq \mathcal{C}$ that best fits our possibly imperfect evidence by jointly minimizing: 1) the number of unexplained tuples; 2) the number of error tuples; and 3) the size of $\mathcal{M}$. The first two are formalized through functions that, for a candidate set $\mathcal{M}$, check how many tuples in the given target instance J (on the right in Figure 2) are unexplained (collectively) by $\mathcal{M}$, and how many tuples resulting from data exchange that are not in J (on the left in Figure 2) are errors for a st tgd in $\mathcal{M}$.

Let $K_\mathcal{C}$ (respectively, $K_\mathcal{M}$ and K_θ) be a canonical universal solution for I and $\mathcal{C}$ (respectively, $\mathcal{M}$ and θ). We consider full st tgds here so canonical universal solutions are unique. Let $\mathsf{creates}_\mathsf{full}(\theta, t)$ be 1 if $t \in K_\theta$ and 0 otherwise. We then define $\mathsf{error}_\mathsf{full}(\mathcal{M}, t)$ for a tuple $t \in K_\mathcal{C} - J$ (Figure 2 left side) to be the number of st tgds in $\mathcal{M}$ for which t is an error.

$$\mathsf{error}_\mathsf{full}(\mathcal{M}, t) = \sum_{\theta \in \mathcal{M}} (\mathsf{creates}_\mathsf{full}(\theta, t)) \tag{1}$$

Correspondingly, for the tuples in J (Figure 2 right side), we define the function $\mathsf{explains}_\mathsf{full}(\mathcal{M}, t)$, which checks whether such a tuple is explained by $\mathcal{M}$.

$$\mathsf{explains}_\mathsf{full}(\mathcal{M}, t) = 1 \text{ if } t \in J \cap K_\mathcal{M} \text{ and 0 otherwise} \tag{2}$$

If a tuple is an error for every candidate (meaning $\mathsf{error}_\mathsf{full}(\mathcal{C}, t) = |\mathcal{C}|$), then we call the tuple a *certain error*. Similarly, tuples in $J - K_\mathcal{C}$ that cannot be explained by any st tgd in $\mathcal{C}$ are called *certain unexplained tuples*. Certain unexplained tuples are depicted in Figure 2(g).

Finally, we define the size function $\mathsf{size}_\mathsf{m}(\mathcal{M})$ to be the sum of the number of atoms in each $\theta \in \mathcal{M}$.

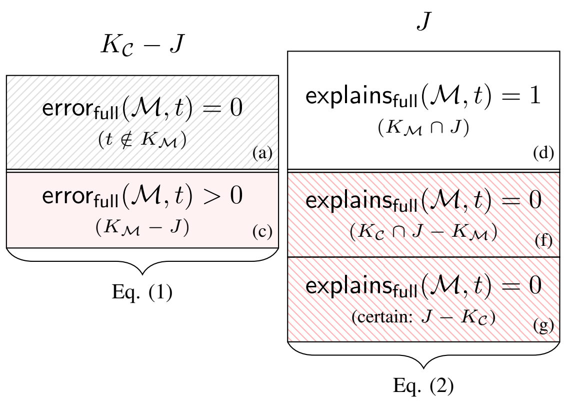
$$\mathsf{size}_\mathsf{m}(\mathcal{M}) = \sum_{\theta \in \mathcal{M}} (\text{number atoms in } \theta) \tag{3}$$

Taking these three criteria together, we formally define the *mapping selection problem for full st tgds* as follows.

Given schemas $\mathbf{S}$, $\mathbf{T}$, a data example (I, J), and a set $\mathcal{C}$ of candidate *full* st tgds

Find
$$\begin{aligned} \underset{\mathcal{M} \subseteq \mathcal{C}}{\operatorname{argmin}}\Big(& \sum_{t \in J} [1 - \mathsf{explains}_\mathsf{full}(\mathcal{M}, t)] \\ & + \sum_{t \in K_\mathcal{C} - J} [\mathsf{error}_\mathsf{full}(\mathcal{M}, t)] \\ & + \mathsf{size}_\mathsf{m}(\mathcal{M})\Big) \end{aligned} \tag{4}$$

The error and size terms of (4) are modular and act as constraints on the supermodular explains term. Such minimization tasks are NP-hard in general, and we have shown that this is also the case for our selection problem [26]. Notice the similarity of the *mapping selection problem* with the formal framework for schema mapping discovery of Gottlob and Senellart [7]. They propose a way of *repairing* a mapping to (1) explain unexplained tuples and to (2) make the mapping valid for an invalid data example (in other words, to account for error tuples). They define an optimal mapping as one that minimizes a cost function containing three parts: the size of the mapping; the number of the repairs needed to account for unexplained tuples; and the number of repairs needed to account for error tuples. Unlike Gottlob and Senellart, we are counting error and unexplained tuples rather than using algebraic operators to repair the mapping. As in Gottlob and Senellart [7], we weight each of these three components equally in our problem definition. However, our formalization permits each rule to



$K_{\mathcal{C}} - J$
J
$\mathsf{error_{full}}(\mathcal{M}, t) = 0$
$(t \notin K_{\mathcal{M}})$
(a)
$\mathsf{error_{full}}(\mathcal{M}, t) > 0$
$(K_{\mathcal{M}} - J)$
(c)
Eq. (1)
$\mathsf{explains_{full}}(\mathcal{M}, t) = 1$
$(K_{\mathcal{M}} \cap J)$
(d)
$\mathsf{explains_{full}}(\mathcal{M}, t) = 0$
$(K_{\mathcal{C}} \cap J - K_{\mathcal{M}})$
(f)
$\mathsf{explains_{full}}(\mathcal{M}, t) = 0$
(certain: $J - K_{\mathcal{C}}$)
(g)
Eq. (2)

Fig. 2: Illustration of functions $\mathsf{error_{full}}$ and $\mathsf{explains_{full}}$.

be weighted differently if there is *a priori* knowledge of the scenarios. For example, if it is known that the target instance contains non-source data, then the penalty for unexplained tuples may be adjusted to be less than for error tuples.

In terms of Figure 2, our goal is to find an $\mathcal{M}$ that jointly minimizes the number of non-certain unexplained tuples (those in (f)), the number of errors (those in (c)) and the size of $\mathcal{M}$. Note that every $\mathcal{M} \subseteq \mathcal{C}$ receives a constant penalty for certain errors (part of (c)) and certain unexplained tuples (the tuples in $J - K_{\mathcal{C}}$ (g)). These tuples can easily be removed for efficiency before running the optimization.

Note a subtle but important difference in how we treat errors and unexplained tuples. The definition of $\mathsf{error_{full}}$ considers each candidate in $\mathcal{M}$ individually, and sums the number of errors made by each. That is, if $\theta_1 \in \mathcal{M}$ and $\theta_2 \in \mathcal{M}$ both make an error on t, that error is counted twice. In other words, we seek a mapping where as few as possible of the st tgds in the mapping make an error on t. In contrast, we do not require each st tgd in the mapping to explain all tuples in J, but consider it sufficient if at least one $\theta \in \mathcal{M}$ explains a tuple. Thus, we cannot treat each θ individually, but we must reason about the set $\mathcal{M}$ as a whole.

IV. Selection over ST TGDs

We now extend our approach to the complete language of st tgds with existentially quantified variables, showing how we assign credit for the shared null values such st tgds introduce. We begin by generalizing our two functions $\mathsf{error_{full}}$ and $\mathsf{explains_{full}}$ to model the partial evidence provided by st tgds with existentials. We then revisit our optimization problem using the new, more general error and $\mathsf{explains}$ functions.

A. Incomplete Errors

In contrast to $\mathsf{error_{full}}$, an error function for arbitrary st tgds has to take into account incomplete tuples, that is, tuples with nulls created by a mapping with existentials.

Example 3: *The candidate θ_1 in Figure 1(d) is not valid with respect to the data example in Figure 1(b). However, if we add the tuple t_1 =task(BigData, Bob, 123) to J then θ_1 is valid for $(I, J \cup t_1)$. But this specific tuple is not in every $J' \supseteq J$ for which θ_1 is valid. Hence, t_1 is not a* full error. *However, a tuple k_1 =task(BigData, Bob, N_0) (where N_0 is a labeled null representing any constant) up to the renaming of the null must be in every such J'. Furthermore, such a tuple is in K_{θ_1}, the canonical universal solution for θ_1 over I.* □

Intuitively, for this example, a tuple in $K_{\mathcal{C}}$ should be an error if there is no homomorphism from that tuple to J. This is sufficient to consider k_1 to be an error for the original J of Figure 1(b), but not an error if we add t_1 to J. However, once an existentially quantified variable is shared between several atoms, we need a more general definition.

Example 4: *The candidate θ_3 in Figure 1(d) is not valid with respect to the extended data example in Figure 1(b)-(c). For it to be valid, J would have to contain two tuples k_1 =task(BigData, Bob, N_0) and k_2 =org(N_0, IBM) with a shared labeled null enabling the join on proj.lead. Suppose we add t_1 =task(BigData, Bob, 123) from above to J and t_2 =org(333, IBM) to J. If we just required each tuple in K_{θ_3} to have a homomorphism to some tuple in J, then neither would be considered an error, as there are homomorphisms from k_1 to t_1 and from k_2 to t_2. However, the instance $J \cup t_1 \cup t_2$ does not correctly connect Bob to IBM. Hence, we would like to consider both tuples to be errors.* □

To address these issues, our error function is based on homomorphisms from all tuples in $K_{\mathcal{C}}$ resulting from a single chase step. If t is in the result of a chase step over $\theta = \forall x\, \phi(x) \rightarrow \exists\, y\, \psi(x, y)$, we call all (target) tuples resulting from this chase step (including t) the *context of t under θ* or *context$_\theta(t)$*.[2] We define the following helper function:

$$\mathsf{creates}(\theta, t) = \begin{cases} 0 & t \in K_{\mathcal{C}} - J,\, t \notin K_\theta \\ 0 & t \in K_\theta - J,\, \exists h : context_\theta(t) \rightarrow J \\ 1 & t \in K_\theta - J \text{ and no such } h \text{ exists} \end{cases} \quad (5)$$

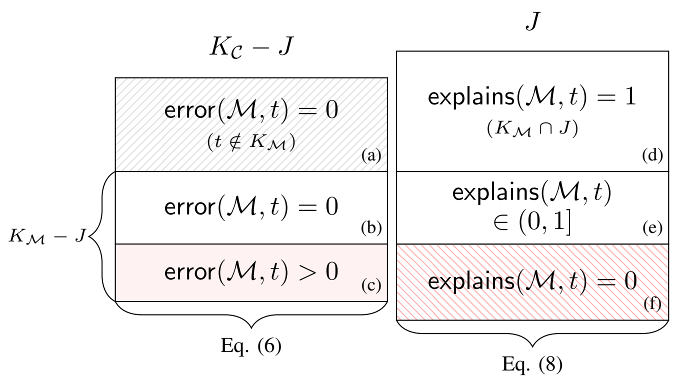
Now for $\mathcal{M} \subseteq \mathcal{C}$ we define the error function as follows.

$$\mathsf{error}(\mathcal{M}, t) = \sum_{\theta \in \mathcal{M}} \mathsf{creates}(\theta, t) \quad (6)$$

In Figure 3, which extends Figure 2 for selection over st tgds, error divides $K_{\mathcal{C}} - J$ into three parts for given $\mathcal{M}$: the tuples in (a) are created by no st tgd in $\mathcal{M}$, those in (b) do not count as errors because they are used to partially explain some tuple in J, and the remaining st tgds in (c) count as errors.

Recall that in the canonical universal instance $K_{\mathcal{C}}$ nulls are only shared between tuples generated by a single chase step. So each incomplete tuple $t \in K_{\mathcal{C}}$ (containing one or more nulls) is associated with a single chase step and st tgd θ. Hence, for such a tuple $\mathsf{error}(t) = 1$ if there is no homomorphism from the $\mathsf{context}_\theta(t)$ to J, and 0 otherwise. For a ground tuple t_g (with no nulls), if there is no homomorphism to J (meaning the tuple is not in J), $\mathsf{error}(t_g)$ is the number of candidate st tgds that create this tuple.

[2]For this to be well-defined, we require that each candidate st tgd θ is normalized into a set of smaller logically equivalent st tgds where only atoms that share existentials are retained in a single st tgd [1].



$K_{\mathcal{C}} - J$
J
$\mathsf{error}(\mathcal{M}, t) = 0$
$(t \notin K_{\mathcal{M}})$
(a)
$\mathsf{explains}(\mathcal{M}, t) = 1$
$(K_{\mathcal{M}} \cap J)$
(d)
$K_{\mathcal{M}} - J$
$\mathsf{error}(\mathcal{M}, t) = 0$
(b)
$\mathsf{explains}(\mathcal{M}, t) \in (0, 1]$
(e)
$\mathsf{error}(\mathcal{M}, t) > 0$
(c)
$\mathsf{explains}(\mathcal{M}, t) = 0$
(f)
Eq. (6)
Eq. (8)

Fig. 3: Illustration of explains and error for selecting st tgds.

B. Partially Explained Tuples

We now extend *explaining* to the case of arbitrary st tgds. More precisely, we use tuples with labeled nulls coming from st tgds with existentially quantified variables to *partially explain* tuples in the target instance J through homomorphisms.

Example 5: *Consider θ_1 in Figure 1(d) and tuple t =task(BigData, Alice, 111) in Figure 1(b). θ_1 partially explains t via a homomorphism from k = task(BigData, Alice, N_1) to t. In the absence of candidates that fully explain t, we might include θ_1 in our selection.* □

To define partial explanation, we treat nulls that play a structural role in connecting information like constants. For a tuple $t \in J$ and a candidate θ, we call $k \in K_\theta$ a *possible explanation* for t under θ if there is a homomorphism $h : context_\theta(k) \rightarrow J$ with $h(k) = t$. Let $\mathsf{E}(t, \theta)$ be the set of all possible explanations for t under θ. We call a labeled null *unique* if it appears exactly once in $context_\theta(k)$. For $k \in \mathsf{E}(t, \theta)$, we define $\mathsf{null}(k)$ to be the number of unique nulls in k divided by the arity of k. So $\mathsf{null}(k) = 0$ if k contains only constants or labeled nulls used at least twice. Otherwise, $\mathsf{null}(k) > 0$. We say that k explains t to degree $1 - \mathsf{null}(k)$, and define the auxiliary function $\mathsf{covers}(\theta, t)$ for $t \in J$ based on the maximal degree to which t is explained by any tuple:

$$\mathsf{covers}(\theta, t) = \begin{cases} \max_{k \in \mathsf{E}(t,\theta)}(1 - \mathsf{null}(k)) & \mathsf{E}(t, \theta) \neq \emptyset \\ 0 & \mathsf{E}(t, \theta) = \emptyset \end{cases} \quad (7)$$

A mapping $\mathcal{M} \subseteq \mathcal{C}$ explains a tuple t as well as the best st tgd $\theta \in \mathcal{M}$ does.

$$\mathsf{explains}(\mathcal{M}, t) = \max_{\theta \in \mathcal{M}} \mathsf{covers}(\theta, t) \quad (8)$$

The function explains can be used to divide J into three parts (Figure 3) for a given $\mathcal{M}$: those tuples fully (d) or partially (e) explained through tuples in (b), and those that cannot be explained by $\mathcal{M}$ at all (f).

Using the same size function as for full st tgds, we define the general *mapping selection problem* as follows:

Given schemas **S**, **T**, a data example (I, J), and a set $\mathcal{C}$ of candidate st tgds

$$\textbf{Find} \quad \underset{\mathcal{M} \subseteq \mathcal{C}}{\operatorname{argmin}} \sum_{t \in J} [(1 - \mathsf{explains}(\mathcal{M}, t))] + \sum_{t \in K_{\mathcal{C}} - J} [\mathsf{error}(\mathcal{M}, t)] + \mathsf{size_m}(\mathcal{M}) \quad (9)$$

The only difference with the case of full st tgds is that we now use notions of error and explaining suitable for st tgds with existentially quantified variables. In terms of Figure 3, we seek a small $\mathcal{M}$ that minimizes the error in (c) and maximizes the degree to which tuples in (d) and (e) are explained.

As in the full case, exactly solving this optimization is NP-hard due to the explains term. In Section V, we provide an efficient approximation algorithm for finding a high quality solution $\mathcal{M}$. An illustrative example of selection over st tgds can be found in the online appendix [26].

V. Probabilistic Mapping Selection

We now introduce Collective Mapping Discovery (**CMD**), our efficient solution for schema mapping selection, using techniques from the field of probabilistic modeling [27] and statistical relational learning (SRL) [28]. Specifically, **CMD** encodes the mapping selection objective (Equation (9)) as a program in probabilistic soft logic (PSL) [14], and uses off the shelf PSL inference to instantiate and solve the optimization problem. Inference in PSL is highly scalable and efficient, as it avoids the combinatorial explosion inherent to relational domains (the relations error and explains) by solving a convex optimization problem, while providing theoretical guarantees on solution quality with respect to the combinatorial optimum.

However, like the majority of SRL methods, PSL relies on a closed world assumption to ensure a well-defined probability distribution. While we will not entirely remove this restriction, we introduce *prioritized disjunctions*, a novel extension to PSL that allows for existentials over closed domains (the existence of an st tgd θ) while maintaining the convexity of inference, which makes it possible to encode and efficiently solve model selection problems such as the mapping selection problem.

A. Probabilistic Soft Logic

PSL [14] is a language for defining collective optimization problems in relational domains. It comes with an efficient and scalable solver for these problems. The key underlying idea is to (1) model desirable properties of the solution as first-order rules, (2) allow random variables to take on soft values between 0 and 1, rather than Boolean values 0 or 1, and (3) let the system find a truth value assignment to all ground atoms in the domain that minimizes the sum of the *distance to satisfaction* of all ground instances of the rules.

A PSL program is a set of weighted rules:

$$w : \quad b_1(\boldsymbol{X}) \wedge \ldots \wedge b_n(\boldsymbol{X}) \rightarrow h_1(\boldsymbol{X}) \vee \ldots \vee h_m(\boldsymbol{X}) \quad (10)$$

where $\boldsymbol{X}$ is a set of universally-quantified variables, the $b_i(\boldsymbol{X})$ and $h_j(\boldsymbol{X})$ are atoms over (subsets of) the variables in $\boldsymbol{X}$, and w is a non-negative weight corresponding to the importance of satisfying the groundings of the rule. In first-order logic, a grounding of such a rule is satisfied if its body evaluates to false (0) or its head evaluates to true (1). PSL generalizes this into a rule's *distance to satisfaction*, which is defined as the difference of the truth values of the body and the head (set to zero if negative), and uses *soft truth values* from the interval $[0,1]$ instead of Boolean ones. It relaxes the logical connectives using the *Lukasiewicz t-norm* and its *co-norm*, which is exact at the extremes, provides a consistent mapping for values in-between, and results in a convex optimization problem. Given an interpretation I of all ground atoms constructed from the predicates and constants in the program, the truth values of formulas are defined as follows.

$$\begin{aligned} I(\ell_1 \wedge \ell_2) &= \max\{0, I(\ell_1) + I(\ell_2) - 1\} \\ I(\ell_1 \vee \ell_2) &= \min\{I(\ell_1) + I(\ell_2), 1\} \\ I(\neg l_1) &= 1 - I(\ell_1) \end{aligned}$$

The distance to satisfaction of a ground rule $r = \text{body} \rightarrow \text{head}$ is defined as follows:

$$d_r(I) = \max\{0, I(\text{body}) - I(\text{head})\} \tag{11}$$

Let R be the set of all ground rules obtained by grounding the program with respect to the given constants. The probability density function f over I is:

$$f(I) = \frac{1}{Z}\text{exp}[-\sum_{r \in R} w_r(d_r(I))] \tag{12}$$

where w_r is the weight of rule r and Z is a normalization constant. PSL inference finds $\text{argmax}_I f(I)$, that is, the interpretation I that minimizes the sum of the distances to satisfaction of all ground rules, each multiplied by the corresponding rule weight. Typically, truth values for some of the ground atoms are provided as evidence, that is, they have *observed* fixed truth values, and we only need to *infer* the optimal interpretation of the remaining atoms. PSL finds an exact optimum using soft truth values, which is then converted to a high quality discrete solution [14].

B. Mapping Selection in PSL

We now encode the mapping selection problem as a PSL program. We introduce three observed predicates that encode tuple membership in the target instance J and the covers and creates functions defined in Section III-C, respectively, and one predicate in whose truth values denote membership of candidate st tgds in the selection, and thus need to be inferred by PSL. A given data example (I, J) and set of candidate st tgds $\mathcal{C}$ will introduce a constant for every tuple in $K_{\mathcal{C}} \cup J$ and for every candidate in $\mathcal{C}$. We use the logical variable F for st tgds, and T for tuples. The **CMD** program consists of the following rules.

$$\begin{aligned} \mathsf{size_m}(F):&\quad in(F) \rightarrow \bot && (13) \\ 1:&\quad J(T) \rightarrow \exists F.\ \mathsf{covers}(F,T) \wedge in(F) && (14) \\ 1:&\quad in(F) \wedge \mathsf{creates}(F,T) \rightarrow J(T) && (15) \end{aligned}$$

Rule (13) implements the size penalty by stating that we prefer not to include an st tgd in the selected set: its weighted distance to satisfaction is $\mathsf{size_m}(f) \cdot (I(in(f)) - 0)$, and thus minimal if $in(f)$ is false. Rule (14) states that if a tuple is in J, there should be an st tgd in the set that covers that tuple, thus implementing the explains term. Note that the existential quantifier is not supported by PSL; we describe how we extend PSL and implement this efficiently in the next subsection. Rule (15) states that if an st tgd creates a tuple, that tuple should be in J, or conversely, if a tuple is not in J (and thus in $K_{\mathcal{C}} - J$), no st tgd in the selected set should create it. This implements the error term of our objective. The advantage of this approach is that it reasons about the interactions between tuples and st tgds in a fine-grained manner. Each rule describes a piece of the mapping selection optimization.

C. Prioritized Disjunction Rules

In first-order logic (with finite domains), formulas with existential quantifiers, such as Rule (14) above, can be rewritten by expanding the existential quantifier into a disjunction over all groundings of its variables; however, in the context of PSL, the resulting disjunction of conjunctions in the head of a rule is expensive and non-convex to optimize in general. We therefore show how to efficiently handle a practically important subclass of such rules through a novel rewriting approach to a collection of regular PSL rules. We call these rules *prioritized disjunction rules*, as they implement a choice among groundings of an existentially quantified variable using observed soft truth values to express preferences or priorities over the alternatives (in the case of Rule (14), over st tgds to be selected). A prioritized disjunction rule is a rule:

$$w:\ b(X) \rightarrow \exists Y.\ h_g(Y,X) \wedge h_i(Y) \tag{16}$$

where $b(X)$ is a conjunction of atoms, $h_g(Y,X)$ is an observed atom and $h_i(Y)$ is an atom whose value will be inferred. The observed truth values of the $h_g(Y,X)$ atoms reflect how good a grounding of Y is for a grounding of X, as the truth value of the head will be higher when assigning high truth values to $h_i(Y)$ with high $h_g(Y,X)$. To efficiently support this comparison of alternatives, we introduce a *k-prioritization* for some natural number k, restricting the truth values of $h_g(Y,X)$ to $\{^0/_k, ..., ^k/_k\}$ only. This allows us to rewrite each prioritized disjunction rule into a collection of rules, where we first expand the existential quantifier in the usual way, and then introduce a rule for each priority level.

Consider first the Boolean case, i.e., $k = 1$. In this case, every disjunct $h_g(Y,X) \wedge h_i(Y)$ is either false or equivalent to $h_i(Y)$. Since $h_g(Y,X)$ is observed, for every grounding y of Y, we can either 1) drop the entire disjunct if $h_g(y,X)$ is false or 2) drop $h_g(y,X)$ if it is true, leaving only $h_i(y)$ in the disjunctive head. This leaves us with a standard PSL rule with a (possibly empty) disjunction of h_i atoms in the head.

For arbitrary k, we generalize this by grouping the head elements based on the priorities. For each grounding $b(x)$ of the rule body $b(X)$, we create one ground rule for every $j = 1, .., k$ of the following form:

$$^w/_k : b(x) \rightarrow \bigvee_{h_g(x,y) \geq {^j/_k}} h_i(y)$$

That is, we have a set of rules with identical bodies whose heads are progressively more general disjunctions of h_i atoms.

$$\begin{aligned}
{}^{w}\!/_{k} : b(x) &\rightarrow \bigvee_{h_g(x,y)\in\{{}^{k}\!/_{k}\}} h_i(y)\\
{}^{w}\!/_{k} : b(x) &\rightarrow \bigvee_{h_g(x,y)\in\{{}^{k}\!/_{k},{}^{(k\,-\,1)}\!/_{k}\}} h_i(y)\\
&\vdots\\
{}^{w}\!/_{k} : b(x) &\rightarrow \bigvee_{h_g(x,y)\in\{{}^{k}\!/_{k},{}^{(k\,-\,1)}\!/_{k},\ldots,{}^{1}\!/_{k}\}} h_i(y)
\end{aligned}$$

To understand the idea behind this transformation, assume for the moment that all $h_i(y)$ have fixed, Boolean truth values, and let ${}^{m}\!/_{k}$ be the highest value $h_g(x,y)$ takes for this x and any y with $h_i(y) = 1$, i.e.,

$${}^{m}\!/_{k} = \max_{\{y|h_i(y)=1\}} h_g(x,y)$$

Then, the rules for $j = 1, .., m$ are satisfied (because their head evaluates to 1), and the ones for $j = (m+1), .., k$ are not satisfied (because their head evaluates to 0). More precisely, their distance to satisfaction is the truth value of $b(x)$, and each of these thus contributes ${}^{w}\!/_{k} \cdot I(b(x))$ to the overall distance to satisfaction, which for this set of ground rules is

$$(k-m)\cdot{}^{w}\!/_{k}\cdot I(b(x)) = w\cdot\left(1-\max_{\{y|h_i(y)=1\}} h_g(x,y)\right)\cdot I(b(x))$$

If b is observed, e.g., $I(b(x)) = 1$ as in the case of (14), this expression depends purely on the maximum value of h_g.

Example 6: *Consider a single grounding of Rule* (14) *for $t =$org(111, SAP) in J from Figure 1(c) and the candidates θ_3 and θ_4 from Figure 1(d). The expanded ground rule is*

$$1 : \;\top \rightarrow \mathsf{covers}(\theta_3, t) \wedge in(\theta_3) \vee \mathsf{covers}(\theta_4, t) \wedge in(\theta_4)$$

Predicate org *has arity two, so we get a* 2*-prioritization with possible values* $\mathsf{covers}(F, t) \in \{{}^{0}\!/_{2}, {}^{1}\!/_{2}, {}^{2}\!/_{2}\}$*. Using values* $\mathsf{covers}(\theta_3, t) = {}^{2}\!/_{2}$ *and* $\mathsf{covers}(\theta_4, t) = {}^{1}\!/_{2}$*, we replace the initial ground rule with*

$$\begin{aligned}
{}^{1}\!/_{2} :&\;\top \rightarrow in(\theta_3) \vee in(\theta_4)\\
{}^{1}\!/_{2} :&\;\top \rightarrow in(\theta_3)
\end{aligned}$$

which completes the rewriting from a rule with existential quantification to a set of regular PSL rules. □

To summarize, we have shown an efficient transformation of a PSL rule with existentials over disjunctions of conjunctions in the head into a (compact) set of regular PSL rules using prioritized disjunctions. Furthermore, the soft-truth value semantics of the disjunction is the maximum over the disjuncts — which we will show to be a useful choice. While this extension was motivated by the mapping selection problem, we expect it to also be useful in other scenarios that involve choices between variable numbers of alternatives.

D. Objective Equivalence

Recall from Equation (9) that our goal is to minimize

$$\sum_{t\in J}[1-\max_{\theta\in\mathcal{M}}\mathsf{covers}(\theta,t)] \tag{17}$$
$$+\sum_{t\in K_{\mathcal{C}}-J}\sum_{\theta\in\mathcal{M}}\mathsf{creates}(\theta,t) \tag{18}$$
$$+\sum_{\theta\in\mathcal{M}}\mathsf{size_m}(\theta) \tag{19}$$

We now demonstrate that, for Boolean values of the $in(\theta)$ atoms, this is exactly the objective used by our PSL program.

We get a grounding of Rule (13) for every st tgd $\theta \in \mathcal{C}$:

$$\mathsf{size_m}(\theta) : in(\theta) \rightarrow \bot \tag{20}$$

For $\theta \in \mathcal{M}$, this rule has distance to satisfaction 1, and 0 else. Thus, each $\theta \in \mathcal{M}$ adds $\mathsf{size_m}(\theta)$ to PSL's distance to satisfaction, so those rules together correspond to (19). The error Rule (15) is trivially satisfied for tuples in J (and any st tgd). Thus, we only need to consider the groundings for $t \in K_{\mathcal{C}} - J$ and $\theta \in \mathcal{C}$:

$$1 : in(\theta) \wedge \mathsf{creates}(\theta, t) \rightarrow \bot \tag{21}$$

Such a ground rule has distance to satisfaction $\mathsf{creates}(\theta, t) - 0 = \mathsf{creates}(\theta, t)$. Recall from Equation (5) that this can only be non-zero for $t \in K_\theta - J$. The PSL sum thus adds $1 \cdot \mathsf{creates}(\theta, t)$ for every $\theta \in \mathcal{M}$ and $t \in K_\theta - J$, which equals (18). Rule (14) is trivially satisfied for $t \notin J$, and for every $t \in J$ results in a partially ground rule

$$1 : \top \rightarrow \exists F.\ \mathsf{covers}(F, t) \wedge in(F) \tag{22}$$

To complete the grounding, we apply PSL's prioritized disjunction rules. Recall (cf. Section IV-B) that the covers function takes on values according to the null function, which is the number of unique nulls divided by the arity of a tuple. Therefore, we know there are values $\{{}^{0}\!/_{k}, \ldots, {}^{k}\!/_{k}\}$ for $\mathsf{covers}(F, t)$ where k is the arity of the tuple t. Thus we get for each $t \in J$ a set of k ground rules, the ith of which is

$$1/k : \;\top \rightarrow \bigvee_{\theta\in\mathcal{C},\mathsf{covers}(\theta,t)\geq{}^{i}\!/_{k}} in(\theta) \tag{23}$$

We know that for every $t \in J$, the associated groundings collectively contribute a distance to satisfaction of

$$1 - 1 \cdot \max_{\theta\in\mathcal{M}} \mathsf{covers}(\theta, t)$$

due to prioritized disjunction rules rewriting, which equals (17). Thus, the **CMD** program optimizes Equation (9).

E. Collective Mapping Discovery

To summarize, given data example (I, J) and candidates $\mathcal{C}$, **CMD** does the following two steps.

1) Compute truth values of evidence from (I, J) and $\mathcal{C}$
2) Perform PSL inference on the **CMD** program and evidence and return the corresponding mapping

Step 1 (*data preparation*) performs data exchange to determine the tuples in $K_{\mathcal{C}}$, and computes the truth values of the $|\mathcal{C}| \times |K_{\mathcal{C}} - J|$ many creates atoms (based on Equation (5)) and of the $|\mathcal{C}| \times |J|$ many covers atoms (based on Equation (7)). While finding a discrete solution to the optimization problem defined by the **CMD** program and evidence is NP-hard, Step 2 (***CMD** optimization*) provides an extremely scalable approximate solution with theoretical quality guarantees.

VI. Evaluation

We experimentally evaluate **CMD** on a variety of scenarios, showing that it robustly handles ambiguous and dirty metadata as well as dirty tuples in the data example, and scales well with the size of the metadata (i.e., schema sizes and number of candidate mappings). We also demonstrate that our prioritized disjunction rules enable efficient inference for realistic complex mapping scenarios. We ran our experiments on an Intel Xeon with 24 x 2.67GHz CPU and 128GB RAM. We provide further details on scenario generation in the online appendix [26]; our implementation of **CMD** and instructions for reproducing all experiments can be found online.[3]

A. Scenario Generation

Each of our scenarios consists of a data example (I, J) for a pair of schemas $\mathbf{S}$ and $\mathbf{T}$ and a set $\mathcal{C}$ of candidate st tgds, which form the input for **CMD**, and a gold standard mapping $\mathcal{M}_G$ used to assess the quality of the solution. Scenario generation is controlled by the parameters listed in Table I and consists of the following five steps.

(1) We generate the initial schemas, correspondences, data example (I, J) and corresponding gold standard mapping $\mathcal{M}_G$ using the metadata generator iBench [18] and data generator ToxGene [29]. iBench uses transformation primitives to create realistic complex mapping scenarios. We chose a representative set of seven primitives (including vertical partitioning, merging relations, etc. [26]) and invoke each of these primitives $\pi_{Invocations}$ times. Three primitives create full mappings, the other four all create existentials used once or twice and include existentials that are shared between relations. These mappings form $\mathcal{M}_G$. Each primitive creates between two and four relations (unless the $\pi_{RelationShare}$ is non-zero meaning a primitive may use a relation created by another primitive rather than creating a new relation). For one invocation, the source schema has eight relations and the target has ten. For ten invocations, we create our largest schemas with eighty source relations and one hundred target relations. The other two parameters controlling this step are $\pi_{TuplesPerTable}$, the number of tuples in each source relation, and $\pi_{RelationShare}$, the percent of source relations that are shared between mappings.

(2) To create dirty metadata, we use iBench to add randomly created foreign keys to π_{FKPerc} percent of the target relations. These foreign keys create erroneous join paths, leading to bad st tgds in our set of candidate mappings.

(3) We add randomly created correspondences to $\pi_{Corresp}$ percent of the target relations [26]. These again add dirty metadata and thus cause bad candidate mappings.

(4) We use the implementation of the Clio [3] algorithm provided by ++Spicy [23] to generate the set of candidates $\mathcal{C}$ from the schemas, foreign key constraints and correspondences generated in previous steps, that is, based on metadata only.

TABLE I: Overview of main experimental parameters.

Parameter	**Range**	**Default**
$\pi_{Invocations}$	1 - 10	2
$\pi_{TuplesPerTable}$	10 - 100	50
$\pi_{RelationShare}$	0 - 40%	0%
π_{FKPerc}	0 - 10%	0%
$\pi_{Corresp}$	0 - 100%	0%
π_{Errors}	0 - 50%	0%
$\pi_{Unexplained}$	0 - 100%	0%

(5) As iBench creates $\mathcal{M}_G$ which is valid and fully explaining for (I, J), we modify J to introduce errors and unexplained tuples (with respect to $\mathcal{M}_G$), where we control the percentages π_{Errors} and $\pi_{Unexplained}$ of non-certain errors and non-certain unexplained tuples in the data example (with respect to the possible maximum), respectively [26].

B. Evaluation of Solution Quality

We assess quality by comparing the mapping $\mathcal{M} \subseteq \mathcal{C}$ selected by **CMD** to (1) the correct mapping $\mathcal{M}_G$ produced by iBench and (2) the set $\mathcal{C}$ of all candidates produced by Clio, which serves as our metadata-only baseline. Directly comparing mappings is a hard problem, so we follow the standard in the literature which is to compare the data exchange solutions produced by the mappings. We use the core data exchange algorithm of ++Spicy [23] to obtain $K_{\mathcal{M}}$ and $K_{\mathcal{C}}$. The gold standard instance K_G for $\mathcal{M}_G$ is the original target instance J obtained from iBench in the first step. We compare these instances using the IQ-Meter [19] quality measure. IQ-Meter measures the ability of a mapping to generate correct tuples as well as correct relational structures via labeled nulls or invented values, so it is appropriate as an evaluation measure for our mappings which contain existentials. It calculates recall and precision of tuples and recall and precision of joins. It also combines these four measures into a distance between zero and one; we use the complement of the distance, which we call $\textsf{IQ-score}(K_1, K_2) \in [0, 1]$, where higher is better.

C. *CMD* Accuracy over Ambiguous Metadata

To assess the performance of **CMD** as the metadata becomes more ambiguous or dirty, we increase the number of candidate st tgds by increasing the π_{FKPerc} parameter from 0 to 10 percent and the $\pi_{Corresp}$ parameter from 0 to 100 percent. We investigate whether the data example allows **CMD** to identify a good mapping from this larger set of candidates. We use four scenarios per parameter setting, with an average of 800 source and 1000 target tuples. Figure 4 shows IQ-scores for Clio (meaning IQ-score(K_G, $K_{\mathcal{C}}$)). Our approach found correct mappings on all problems, while the IQ-score for the $\mathcal{C}$ baseline decreased with more imprecise evidence, thus showing that **CMD** effectively handles ambiguous metadata.

D. *CMD* Accuracy over Dirty Data

Our second experiment investigates the effect of imperfect data on mapping quality. We set $\pi_{Corresp} = 100$ percent to increase the number of potential non-certain unexplained tuples, and vary the percentage π_{Errors} of non-certain errors

[3] http://projects.linqs.org/project/cmd

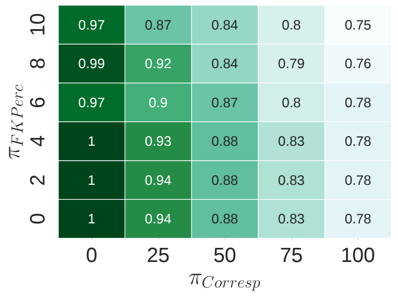


Fig. 4: Mapping quality (IQ-Score) for the Clio baseline with ambiguous metadata. **CMD** always reaches IQ-Score 1.

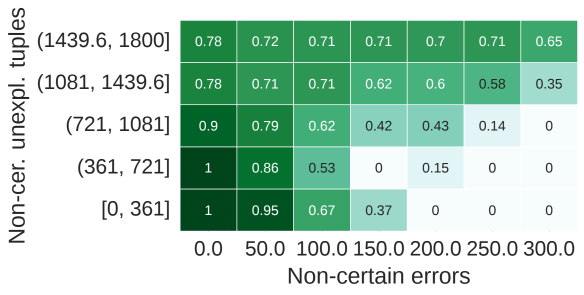


Fig. 5: Mapping quality (IQ-Score) for **CMD** with dirty data.

from 0 to 30% in steps of five, and the percentage $\pi_{Unexplained}$ of non-certain unexplained tuples from 0 to 100% in steps of 25. We consider three scenarios in each case, with 800 source tuples and 1000 tuples in the initial target instance J. The numbers of non-certain errors and unexplained tuples obtained range from zero to 300 and zero to 1800, respectively.

In Figure 5, we group scenarios based on the (exact) number of errors, and based on similar numbers of unexplained tuples and plot average IQ Score for each group. Generally, as the number of errors increases, the quality of the mapping selected by **CMD** decreases. Adding non-certain unexplained tuples also generally decreases the quality of the mapping. However, in the presence of significant numbers of errors unexplained tuples increase mapping quality. This happens whenever unexplained tuples cause **CMD** to select st tgds in $\mathcal{C}-\mathcal{M}_G$ that are similar to st tgds in $\mathcal{M}_G$, and when selecting those st tgds is preferred over the empty mapping. For scenarios with over one quarter unexplained tuples, and even in the presence of a few (less than 10%) errors, **CMD** routinely finds mappings with high IQ scores.

E. *Performance of **CMD***

The next set of experiments evaluates the performance of our approach along several dimensions. We focus on *optimization time*, i.e., the time to find an optimal mapping after data preparation is completed. Data preparation (determining which tuples are errors or unexplained for each st tgd in $\mathcal{C}$) is slow, taking up to 150 minutes in our largest scenarios, and we have not studied any optimizations for it. It can be done in batch overnight. Furthermore, it is modular and thus could be parallelized, and easily updated for changes to $\mathcal{C}$.

First, we vary $\pi_{TuplesPerTable}$ from 10 to 100 in steps of 10 tuples to obtain data examples of increasing size for our default schema size of 36 relations. We generate five scenarios for each setting. Figure 6(a) plots average optimization time in **CMD** and average time to generate $\mathcal{C}$ (the Clio baseline). **CMD** optimization times are comparable to Clio times (note this is the ++Spicy implementation of Clio) even though we optimize over relatively large (3600 tuple) data examples.

Second, we vary $\pi_{Invocations}$ from 1 to 10 to increase the sizes of the schemas (and thus the number of candidate st tgds that are plausible for the schemas), which results in source and target schemas with 18 to 180 total relations. The largest scenario involved 70 candidate st tgds, or over 10^{22} possible mappings. We set $\pi_{TuplesPerTable} = 50$, thus obtaining data examples with 900 to 9000 tuples. We use five scenarios per setting and, as before, plot average **CMD** optimization time and average time to run Clio in Figure 6(b), where the horizontal axis shows the total number of relations (split into six bins of equal size). Again, **CMD** optimization times are comparable to those of the Clio implementation for generating $\mathcal{C}$, but for increasing schema size, the latter increases more quickly.

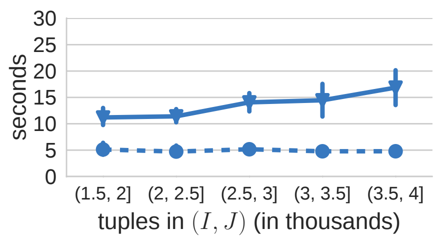
Third, we consider scenarios with the relation sharing parameter $\pi_{RelationShare}$ set to $0, 15, 30, 35$ and 40%, which represents realistic mappings [18]. This increases the number of relations that are used in more than one candidate st tgd, and thereby (indirectly) the number of st tgds that may explain each tuple, and thus the complexity of the expensive explains terms in our objective. We generate five scenarios per setting, all with 20 target relations, 8-16 source relations and up to 3600 tuples (100 per relation). The number of st tgds that (partially) explain a target tuple within each scenario varies from zero to five. In Figure 6(c), we plot average optimization times of **CMD** and the running times of Clio, where the horizontal axis is the maximum number of candidates that explain each tuple in each scenario (no scenario had a maximum of four). These results confirm that at realistic levels of relation sharing, **CMD** optimization still selects a mapping quickly.

F. *CMD on Real Metadata and Data*

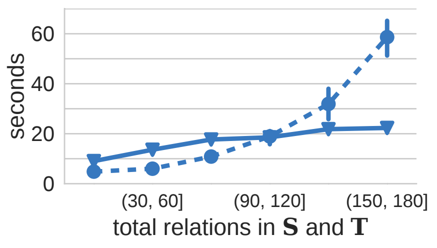
Finally, we consider a simple scenario with real schemas and data from the well-known Amalgam benchmark [30]. We use a scenario setting Amalgam schema *A1* as source and *A3* as target. To construct a data example, we select a small subset of the data in A1. To test whether **CMD** handles dirty metadata, we include random correspondences on one third of the target relations. The final evaluation contains 18 relations and 2,502 tuples. For this scenario, **CMD** achieves a seven percent improvement in mapping quality (IQ-Score) over the Clio baseline. While preliminary, these results are promising.

VII. Related Work

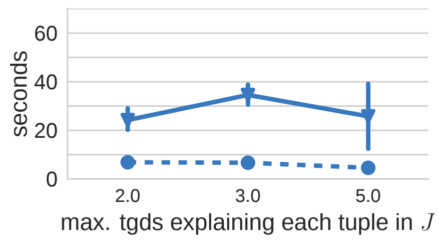
Using Metadata. Using metadata information to guide schema mapping discovery has a long tradition. The names of schema elements (such as attributes) can be used to suggest attribute correspondences (the well-known schema matching problem) and the Clio project showed how the schemas and constraints can be used to infer mappings [2], [3]. HepTox [25] and ++Spicy [23] have extended this to richer forms of metadata (including equality-generating-dependencies). In addition, the



(a)



(b)



(c)

Fig. 6: Optimization time of **CMD** (solid) and time to run Clio (dashed) w.r.t. size of (a) data example, (b) schemas, and (c) largest set of st tgds explaining the same tuple.

role of data in resolving ambiguity or incompleteness in the metadata evidence has long been recognized, both in matching [31] and in schema mapping, where the Data Viewer [20] and Muse [5] systems use source and target data interactively to help a user understand, refine or correct automatically generated mappings. Most systems that combine evidence from metadata and data, do so heuristically and may fail to suggest a good mapping under inconsistent evidence.

Using Data Examples. A complementary approach that uses data only is often called example-driven schema-mapping design [32]. An example that is closest to ours casts schema mapping discovery from data as a formal (and fully automated) learning problem [7]. Given a single data example (I, J) find a mapping M that is valid and fully explaining (and of minimal size) for (I, J). Even for full st tgds finding optimal mappings in this framework is DP-hard [7]. Using this framework, ten Cate et al. [11] consider the restricted class of mappings with a single atom (relation) in the head and in the body. They provide a greedy approximation algorithm that is guaranteed to find near optimal (valid and fully explaining) mappings of this type, but do not discuss experimental results.

In contrast, we do not require the mapping to be valid or fully explaining, rather we define an optimization problem that finds an optimal set of st tgds that minimizes errors (the invalidity of a mapping) and unexplained tuples. Although the number of errors can, in theory, be exponential in the size of the mappings (as pointed out by Gottlob and Senellart [7]), we manage this complexity by using a set of candidate st tgds derived from real mapping discovery systems and by using an efficient approximation framework (PSL) to reason over these alternative mappings. We also provide a novel principled way of combining evidence from mappings that contain existentials (and hence only partially explain tuples).

Multiple Examples. The Eirene system returns the most general mapping that fits a set of data examples, if one exists, and otherwise guides a user in refining her data (not the mapping) to identify tuples that are causing the failure [21]. This is in contrast to Muse and the Data Viewer systems that interactively pick data to help a user refine a mapping. Alexe et al. [12] have also studied when a mapping can be uniquely characterized by a set of data examples. This problem has also been cast as a learning problem, where a user labels a series of examples as positive or negative [33]. Finally, Sarma et al. [34] consider how to learn **views** (mappings with a single relational atom and no existentials in the target) from data alone.

Related Selection Problems. Belhajjame et al. [13] use feedback from users on exchange solutions to estimate the precision and recall of views (or full GAV mappings). They present view selection as an optimization problem that maximizes either precision (which is maximal for valid mappings) or recall (which is maximal for fully explaining mappings), without taking mapping size into account. While Belhajjame et al. [13] do not provide runtimes for their approach, they use a very powerful, general purpose search algorithm [35] designed for constrained optimization problems. In contrast, our mapping selection problem, though NP-hard, is of a form for which PSL efficiently finds a high quality solution.

While our approach relies on a potentially noisy data example (I, J) to select among mappings, Belhajjame et al. [13] rely on potentially noisy feedback from a user, who annotates target tuples in a query answer as expected (with respect to an implicit J) or unexpected, or provides additional expected tuples. User feedback has also been used in active learning scenarios in the context of data integration, e.g., to select consistent sets of attribute-attribute matches among many datasources [36], or to select join associations in the context of keyword-search based data integration [37]. While those settings are quite different from the one we consider here, extending **CMD** with active learning to incorporate additional feedback is an interesting direction for future work.

Similarly, the source selection problem [38] has been modeled as a problem of finding a set of sources that minimize the cost of data integration while maximizing the gain (a score that is similar to recall). Dong et al. [38] use the greedy randomized adaptive search procedure meta-heuristic to solve the source selection problem, a heuristic which unlike PSL does not provide any approximation guarantees on the solution.

Probabilistic Reasoning. Statistical relational techniques have been applied to a variety of data and knowledge integration problems. Perhaps closest to our approach is the use of Markov Logic [39] for ontology alignment [40] and ontological mapping construction [41]. However, we consider more expressive mappings than either of those approaches. Furthermore, by using PSL, we can easily integrate partial evidence from st tgds with existential quantification through soft truth values. More importantly, in contrast to Markov Logic, PSL avoids the hard combinatorial optimization problem and instead provides scalable inference with guarantees on solution quality. This advantage has proven crucial also for applications of PSL in knowledge graph identification [15] and data fusion [16], [17].

VIII. Conclusion

We introduce *Collective Mapping Discovery* (**CMD**), a new approach to schema mapping selection that finds a set of st tgds that best explains the data in the sources being integrated. We use both metadata and data as evidence to resolve ambiguities and incompleteness in the sources, allowing some inconsistencies and choosing a small set of mappings that work collectively to explain the data. To solve this problem, we use and extend probabilistic soft logic (PSL), casting the problem as efficient joint probabilistic inference. The declarative nature of the PSL program makes it easy to extend **CMD** to include additional forms of evidence and constraints, coming from the domain, from user feedback, or other sources. In future work, we plan to explore weight learning techniques and investigate their impact in different problem settings.

Acknowledgments: AK is supported by the Research Foundation Flanders (FWO), RJM by NSERC, and LG by the National Science Foundation [IIS1218488]. The authors thank the anonymous reviewers for their helpful feedback; and Boris Glavic, Patricia Arocena, Gianni Mecca, Donatello Santoro, and Radu Ciucanu for valuable help with iBench and ++Spicy.

References

[1] B. ten Cate and P. G. Kolaitis, "Structural characterizations of schema-mapping languages," *Commun. ACM*, vol. 53, no. 1, pp. 101–110, 2010.

[2] R. J. Miller, L. M. Haas, and M. Hernández, "Schema Mapping as Query Discovery," in *VLDB*, 2000.

[3] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin, "Translating Web Data," in *VLDB*, 2002.

[4] H. Elmeleegy, A. K. Elmagarmid, and J. Lee, "Leveraging query logs for schema mapping generation in U-MAP," in *SIGMOD*, 2011.

[5] B. Alexe, L. Chiticariu, R. J. Miller, and W.-C. Tan, "Muse: Mapping understanding and design by example," in *ICDE*, 2008.

[6] B. ten Cate, P. G. Kolaitis, and W.-C. Tan, "Schema mappings and data examples," in *EDBT*, 2013.

[7] G. Gottlob and P. Senellart, "Schema mapping discovery from data instances," *Journal of the ACM (JACM)*, vol. 57, no. 2, p. 6, 2010.

[8] L. Qian, M. J. Cafarella, and H. Jagadish, "Sample-driven schema mapping," in *SIGMOD*, 2012.

[9] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa, "Schema mapping verification: the spicy way," in *EDBT*, 2008.

[10] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan, "Designing and refining schema mappings via data examples," in *SIGMOD*, 2011.

[11] B. ten Cate, P. G. Kolaitis, K. Qian, and W.-C. Tan, "Approximation algorithms for schema-mapping discovery from data examples," in *AMW*, 2015.

[12] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan, "Characterizing schema mappings via data examples," *ACM Trans. Database Syst.*, vol. 36, no. 4, p. 23, 2011.

[13] K. Belhajjame, N. W. Paton, S. Embury, A. A. Fernandes, and C. Hedeler, "Incrementally improving dataspaces based on user feedback," *Information Systems*, 2013.

[14] S. H. Bach, M. Broecheler, B. Huang, and L. Getoor, "Hinge-loss markov random fields and probabilistic soft logic," *ArXiv:1505.04406 [cs.LG]*, 2015.

[15] J. Pujara, H. Miao, L. Getoor, and W. Cohen, "Knowledge graph identification," in *ISWC*, 2013.

[16] S. Fakhraei, B. Huang, L. Raschid, and L. Getoor, "Network-based drug-target interaction prediction with probabilistic soft logic," *IEEE/ACM Trans. Comput. Biology Bioinform.*, vol. 11, no. 5, pp. 775–787, 2014.

[17] P. Kouki, S. Fakhraei, J. Foulds, M. Eirinaki, and L. Getoor, "HyPER: A flexible and extensible probabilistic framework for hybrid recommender systems," in *RecSys*, 2015.

[18] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller, "The iBench Integration Metadata Generator," *PVLDB*, vol. 9, no. 3, pp. 108–119, 2015.

[19] G. Mecca, P. Papotti, and D. Santoro, "IQ-METER - an evaluation tool for data-transformation systems," in *ICDE*, 2014.

[20] L.-L. Yan, R. J. Miller, L. Haas, and R. Fagin, "Data-Driven Understanding and Refinement of Schema Mappings," in *SIGMOD*, 2001.

[21] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan, "EIRENE: Interactive design and refinement of schema mappings via data examples," *PVLDB*, vol. 4, no. 12, pp. 1414–1417, 2011.

[22] 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: Foundations and Applications - Essays in Honor of John Mylopoulos*, 2009, pp. 198–236.

[23] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro, "++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange," *PVLDB*, vol. 4, no. 12, pp. 1438–1441, 2011.

[24] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa, "Data exchange: semantics and query answering," *Theor. Comput. Sci.*, vol. 336, no. 1, pp. 89–124, 2005.

[25] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger, "HePToX: Marrying XML and Heterogeneity in Your P2P Databases," in *VLDB*, 2005.

[26] A. Kimmig, A. Memory, R. J. Miller, and L. Getoor, "A collective, probabilistic approach to schema mapping: Appendix," *ArXiv:1702.03447 [cs.DB]*, 2017.

[27] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[28] L. Getoor and B. Taskar, Eds., *An Introduction to Statistical Relational Learning*. MIT Press, 2007.

[29] D. Barbosa, A. O. Mendelzon, J. Keenleyside, and K. A. Lyons, "Toxgene: a template-based data generator for XML," in *SIGMOD*, 2002.

[30] R. J. Miller, D. Fisla, M. Huang, D. Kymlicka, F. Ku, and V. Lee, "The Amalgam Schema and Data Integration Test Suite," 2001, www.cs.toronto.edu/~miller/amalgam.

[31] J. Kang and J. F. Naughton, "On schema matching with opaque column names and data values," in *SIGMOD*, 2003.

[32] B. ten Cate, P. G. Kolaitis, and W. C. Tan, "Schema mappings and data examples," in *EDBT*, 2013.

[33] B. ten Cate, V. Dalmau, and P. G. Kolaitis, "Learning schema mappings," *ACM Trans. Database Syst.*, vol. 38, no. 4, p. 28, 2013.

[34] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom, "Synthesizing view definitions from data," in *ICDT*, 2010.

[35] C. Audet and J. E. Dennis Jr., "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 188–217, 2006.

[36] N. Q. V. Hung, N. T. Tam, Z. Miklós, K. Aberer, A. Gal, and M. Weidlich, "Pay-as-you-go reconciliation in schema matching networks," in *ICDE*, 2014.

[37] Z. Yan, N. Zheng, Z. G. Ives, P. P. Talukdar, and C. Yu, "Actively soliciting feedback for query answers in keyword search-based data integration," *PVLDB*, vol. 6, no. 3, pp. 205–216, 2013.

[38] X. L. Dong, B. Saha, and D. Srivastava, "Less is more: Selecting sources wisely for integration," *PVLDB*, vol. 6, no. 2, pp. 37–48, 2012.

[39] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[40] M. Niepert, C. Meilicke, and H. Stuckenschmidt, "A probabilistic-logical framework for ontology matching," in *AAAI*, 2010.

[41] C. Zhang, R. Hoffmann, and D. S. Weld, "Ontological smoothing for relation extraction with minimal supervision," in *AAAI*, 2012.