A Collective, Probabilistic Approach to Schema Mapping Using Diverse Noisy Evidence

Angelika Kimmig, Alex Memory, Member, IEEE, Renée J. Miller, Member, IEEE, and Lise Getoor, Member, IEEE

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 schema 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 as well as inconsistencies and incompleteness in the input. We then introduce Collective Mapping Discovery (CMD), our solution to this problem using state-of-the-art probabilistic reasoning techniques. Our evaluation on a wide range of integration scenarios, including several real-world domains, demonstrates that CMD effectively combines data and metadata information to infer highly accurate mappings even with significant levels of noise.

1 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 prove that exactly solving this problem is NP-hard already for full st tgds, i.e., st tgds without existential quantifiers. We then provide an efficient and highly accurate approximate 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]. It however did not support the kind of open world reasoning required for mapping selection, where we need to express constraints over the existence of elements in a set satisfying certain conditions, namely, st tgd in the mapping explaining tuples in the data example, and furthermore, preferences over these elements are available. We therefore extend PSL with prioritized disjunctions, which provide a tractable framework for handling such existential, weighted constraints, and thereby allow us to define key features of the mapping selection problem. To use data and metadata as input, we use the extended PSL language as a common representation for both. The data evidence comprises a data example and the metadata evidence comprises a set of st tgd. By having a common language for reasoning, we can easily integrate data and metadata evidence by, for example, reasoning about whether a data example satisfies metadata evidence such as part of a mapping.

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 tgd. CMD advances the state-of-the-art in schema mapping by using more kinds of evidence and integrating them 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. 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. We demonstrate that the approach scales well with the size of both metadata and data, and effectively selects small, correct mappings even if dozens of competing candidate mappings are available for each tuple. In addition, we show that CMD is effective on several problems with real data.

This paper expands on our earlier work [20], proving complexity results, and contributing experimental results on additional real world problems as well as problems with an order of magnitude greater complexity. Section 2 illustrates the key challenges with an example. Section 3 introduces the selection problem for st tgd without existentially quantified variables, and Section 4 extends this to st tgd. Section 5 introduces our solution using PSL and our extension of PSL with prioritized disjunctions. We discuss experiments in

Section 6 and related work in Section 7.

2 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

![Diagram](image-url)
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).

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 $\theta_0$ and $\theta_1$. Notice that the data example is valid for $\theta_0$ (meaning (I)) satisfy the mapping $\theta_0$ but is not valid with respect to $\theta_1$, 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 Alice is wrong and should be Bob) causing this error. If the data is clean, this error tuple would suggest that we should prefer $\theta_0$ over $\theta_1$.

Note that $\theta_0$ and $\theta_1$ both ignore the correspondence between emp.company and org.name. Mapping $\theta_2$ 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 $\theta_2$ and $\theta_3$. The mapping $\theta_2$ alone maps the inner join of the source data to the target. Mappings $\theta_2$ and $\theta_4$ together map the right outer-join.

If we also add the leader tuples in Figure 1(c) to our data example, $\theta_5$ explains all leader tuples. However, $\theta_5$ 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 $\theta_5$. The mapping $\theta_5$ 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 $\{\theta_2, \theta_4, \theta_0\}$ is a good choice.

Note that our candidates $\theta_0 - \theta_6$ use only correspondences $c_1$-$c_4$ in Figure 1(a). If a matcher incorrectly suggested correspondence $c_5$, then we may get additional candidate mappings like $\theta_7$ or $\theta_8$ 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 ($\theta_0$) and some that are consistent with a join on lead ($\theta_1$); e.g., (ML, Alice, 111) is consistent with both $\theta_0$ and $\theta_1$. 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 fail in the presence of such unexplained tuples. 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 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.

**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 $\theta_0$ and $\theta_1$ 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 $\theta_2$ over the combination of $\theta_5$ and $\theta_4$. 

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.
since the data supports the connection $\theta_2$ 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 extends evidence. to collectively minimize the number of errors and number of unexplained tuples, even under contradictory or incomplete evidence.

3 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 4.

3.1 Mapping Selection Inputs

We define our mapping selection problem with respect to a source schema $S$ and a target schema $T$, where a schema is a set of relations. The data evidence consists of a data example, that is, a pair of instances $I$ of $S$ and $J$ of $T$. The metadata evidence consists of a (finite) set $C$ of candidate st tgds. An st tgd is a logical formula $\forall x \phi(x) \rightarrow \exists y \psi(x, y)$, where $\phi$ is a conjunction of atoms over the relations of $S$ and $\psi$ over those of $T$ [1]. Here, $x$ and $y$ are sets of logical variables. If $y$ is empty (no existentials) then the st tgd is a full st tgd [25].

Candidate st tgds can be generated using existing schema mapping systems. Such systems, both industrial systems 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 [23], HepTox [26], and ++Spicy [24] 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.

3.2 Characterizing the Input Quality

Given metadata evidence $(S, T, C)$, our goal is to find a subset $M \subseteq C$ that best “fits” the data example $(I, J)$. Let $C$ be the set of all constants in $I \cup J$, and $N$ a set of labeled nulls (disjoint from $C$). Following Fagin et al. [25], a homomorphism between instances $h : K_1 \rightarrow K_2$ is a mapping from $C \cup N$ to $C \cup N$ such that: (1) for every $c \in C$, $h(c) = c$, 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 $C \cup N$ such that for every $R(x_1, \ldots, x_n)$ in $\phi(x)$, $R(h(x_1), \ldots, h(x_n))$ is in $K$. Let $M$ be a set of st tgds, then an instance $K$ of $T$ is called a solution for $I$, if $(I, K) \models M$. 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. [25] showed how a universal solution can be computed efficiently using the chase over $M$ (and such a universal solution is typically called a canonical universal solution).

Gottlob and Senellart [7] call a mapping $M$ valid for $(I, J)$ if $J$ is a solution for $I$ under $M$. Suppose $(I, J) \not\models M$. 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 $J$ and in every $J'$ such that $(I, J \cup J') \models M$. If $K$ is a universal solution for $M$ and $I$, then $t$ is a full error iff $t \in K$ and $t \not\in J$. If $(I, J)$ is valid with respect to $M$ then there are no full errors.

Example 1: The candidate $\theta_3$ 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 $\theta_3$ 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 $M \subseteq C$ and source instance $I$ explain a ground fact $t$, if $t \in K$ for every $K$ such that $(I, K) \models M$. A mapping $M$ and $I$ fully explain $J$ if they explain every tuple in $J$. A ground tuple $t$ is explained by $M$ and $I$ iff $t$ is in a universal solution for $M$ and $I$.

Example 2: Consider again $\theta_3$ of Figure 1(d). For the instance $I$ of $emp$ shown in (b), $\theta_3$ fully explains $J$ (the two leader tuples) shown in (c). However, if leader also contained leader(Joe), then $\theta_3$ would still be valid, but leader(Joe) is an unexplained tuple.

3.3 Collective Selection over Full Mappings

We define an optimization problem for finding a mapping $M \subseteq C$ that best fits our imperfect evidence by jointly minimizing:

1) the number of unexplained tuples;
2) the number of error tuples; and
3) the size of $M$.

The first two are formalized through functions that, for a candidate set $M$, compare the given target instance $J$ to the solution for $M$ and $I$, i.e., check how many tuples in $J$ are unexplained (collectively) by $M$, and how many tuples resulting from data exchange with each st tgd in $M$ are not in $J$. Figure 2 illustrates these different kinds of tuples.

Let $K_C$ (respectively, $K_M$ and $K_\theta$) be a canonical universal solution for $I$ and $C$ (respectively, $M$ and $\theta$). We consider full st tgds so canonical universal solutions are unique. We define $create_{full}(\theta, t)$ as follows:

$$create_{full}(\theta, t) = \begin{cases} 1 & t \in K_\theta \\ 0 & \text{otherwise} \end{cases}$$ (1)

We then define $error_{full}(M, t)$ for a tuple $t \in K_C - J$ (Figure 2 left side) to be the number of st tgds in $M$ for which $t$ is an error.

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

\[
\text{explains}_{\text{full}}(M, t) = 1 \text{ if } t \in J \cap K_C \text{ and } 0 \text{ otherwise} \quad (3)
\]

We call tuples in $J - K_C$ that cannot be explained by any st tgds in $C$ unexplainable tuples (Figure 2(g)).

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

\[
\text{size}(M) = \sum_{\theta \in M} (\text{number atoms in } \theta) \quad (4)
\]

This choice of complexity term provides a guard against including st tgds with insufficient support from the evidence. Taking these three criteria together, we formally define the mapping selection problem for full st tgds as follows.

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

**Find** $\arg\min_{M \subseteq C} \left[ \sum_{t \in J} [1 - \text{explains}_{\text{full}}(M, t)] + \sum_{t \in K_C - J} \text{error}_{\text{full}}(M, t) + \text{size}(M) \right] \quad (5)$

As we show in Section 3.4, this problem is NP-hard.

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. In contrast, we are counting error and unexplained tuples rather than using algebraic operators to repair the mapping. We weight each of these three components equally in our problem definition. However, our formalization permits each part to be weighted differently if there is a priori knowledge of the scenarios.

In terms of Figure 2, our goal is to find an $M$ that jointly minimizes the number of unexplained but explainable tuples (those in (f)), the number of errors (those in (c)) and the size of $M$. Note that every $M \subseteq C$ receives a constant penalty for unexplainable tuples (the tuples in $J - K_C$ (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 $\text{error}_{\text{full}}(\cdot)$ considers each candidate in $M$ individually, and sums the number of errors made by each. That is, if two st tgds $\theta_i \in M$ and $\theta_j \in M$ 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 $\theta \in M$ explains a tuple. Thus, we cannot treat each $\theta$ individually, but we must reason about the set $M$ as a whole.

### 3.4 Mapping Selection is NP-hard

The $\text{error}_{\text{full}}(\cdot)$ and $\text{size}(\cdot)$ terms of (5) are modular and act as constraints on the supermodular $\text{explains}_{\text{full}}(\cdot)$ term. Such minimization tasks are often NP-hard, and we provide proof that this is also the case for our selection problem.

**Theorem 1.** The mapping selection problem for full st tgds as defined in (5) is NP-hard.

**Proof.** We use a reduction from SET COVER, which is well known to be NP-complete, and is defined as follows:

**Given** a finite set $U$, a finite collection $R = \{R_i \mid R_i \subseteq U, 1 \leq i \leq k\}$ and a natural number $n \leq k$, is there a set $R' \subseteq R$ consisting of at most $n$ sets $R_i$ such that $\bigcup_{R_i \in R'} R_i = U$?

We first consider the decision variant of mapping selection, which is defined as follows:

**Given** schemas $S$, $T$, a data example $(I, J)$, a set $C$ of candidate full st tgds, and a natural number $m$, is there a selection $M \subseteq C$ with $F(M) \leq m$?

where $F(M)$ is the function minimized in (5), i.e.,

\[
F(M) = \sum_{t \in J} [1 - \text{explains}_{\text{full}}(M, t)] + \sum_{t \in K_C - J} \text{error}_{\text{full}}(M, t) + \text{size}(M) \quad (6)
\]

We construct a mapping selection decision instance from a SET COVER instance as follows. We set $m = 2n$, introduce an auxiliary domain $D = \{1, \ldots, m + 1\}$, and define

- $S = \{R_i/2 \mid R_i \in R\}$
- $T = \{U/2\}$
- $C = \{R_i(X, Y) \rightarrow U(X, Y) \mid R_i \in R\}$
- $J = \{U(x, y) \mid (x, y) \in U \times D\}$
- $I = \bigcup_{R_i \in R} \{R_i(x, y) \mid (x, y) \in R_i \times D\}$

We use notation $R/k$ to indicate relation $R$ has arity $k$. It is easily verified that this construction is polynomial in the size of the SET COVER instance. It is easily verified that this construction is polynomial in the size of the SET COVER instance. We next show that the answers to SET COVER and the constructed mapping selection problem coincide.
For each $R_i$, the candidate st tgd $\theta_i = R_i(X, Y) \rightarrow U(X, Y)$ has size two, makes no errors (as $R_i \subseteq U$), and for each $x \in R_i$ explains the tuples $U(x, 1), \ldots, U(x, m + 1)$. We thus have

$$F(M) = \sum_{t \in J} [1 - \text{explains}_{\theta_i}(M, t)] + 2 \cdot |M|$$

(7)

$$= (m + 1) \cdot (|U| - |\bigcup_{t \in M} R_i|) + 2 \cdot |M|$$

(8)

A mapping $M \subseteq C$ with $F(M) \leq m = 2n$ thus exists if and only if $|\bigcup_{t \in M} R_i| = |U|$ and $|M| \leq n$, which is exactly the case where $\mathcal{M}$ encodes a covering selection with at most $n$ sets. Furthermore, if such mappings exist, the optimal mapping according to (5) is one of them, and a polynomial time solution for mapping selection with full st tgds can thus be used to find a candidate solution that can be verified or rejected in polynomial time to answer SET COVER.

4 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 $\text{error}_{\theta_i}(\cdot)$ and $\text{explains}_{\theta_i}(\cdot)$ to model the partial evidence provided by st tgds with existentials. We then revisit our optimization problem using the new, more general functions.

4.1 Incomplete Errors

In contrast to $\text{error}_{\theta_i}(\cdot)$, 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 $\theta_1$ in Figure 1(d) is not valid with respect to the data example in Figure 1(b). However, if we add the tuple $t_1 = \text{task}(\text{BigData, Bob, 123})$ to $J$ then $\theta_1$ is valid for $(I, J \cup t_1)$. But this specific tuple is not in every $J' \supseteq J$ for which $\theta_1$ is valid. Hence, $t_1$ is not a full error. However, a tuple $k_1 = \text{task}(\text{BigData, Bob, N_0})$ (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 $K_{\theta_1}$, the canonical universal solution for $\theta_1$ over $I$.

Intuitively, for this example, a tuple in $K_C$ 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 $\theta_3$ 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 $k_1 = \text{task}(\text{BigData, Bob, N_0})$ and $k_2 = \text{org}(N_0, \text{IBM})$ with a shared labeled null enabling the join on $\text{proj}$. Suppose we add $t_1 = \text{task}(\text{BigData, Bob, 123})$ from above to $J$ and $t_2 = \text{org}(333, \text{IBM})$ to $J$. If we just required each tuple in $K_{\theta_3}$ 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($\cdot$) function is based on homomorphisms from all tuples in $K_C$ 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 $\theta$ or context$\theta(t)$.

We define the following helper function:

$$\text{creates}(\theta, t) = \begin{cases} 0 & t \in K_C - J, t \notin K_{\theta} \\ 1 & t \in K_{\theta} - J \text{ or no such } h \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

(9)

Now for $M \subseteq C$ we define the error($\cdot$) function as follows.

$$\text{error}(M, t) = \sum_{\theta \in M} \text{creates}(\theta, t)$$

(10)

In Figure 3, which extends Figure 2 for selection over st tgds, error($\cdot$) divides $K_C - J$ into three parts for given $M$: the tuples in (a) are created by no st tgd in $M$, those in (b) do not count as errors because homomorphisms exist from them to $J$, and the remaining st tgds in (c) count as errors.

Recall that in the canonical universal instance $K_C$ nulls are only shared between tuples generated by a single chase step. So each incomplete tuple $t \in K_C$ (containing one or more nulls) is associated with a single chase step and st tgd $\theta$. Hence, for such a tuple $t$, Equation (10) evaluates to 1 if there is no homomorphism from the context$\theta(t)$ to $J$, i.e., $t$ is an error, 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 $J$), Equation (10) counts how many candidates make this error.

4.2 Partially Explained Tuples

We now extend explaining to 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 $\theta_1$ in Figure 1(d) and tuple $t = \text{task}(\text{BigData, Alice, 111})$ in Figure 1(b). $\theta_1$ partially explains $t$ via a homomorphism from $k = \text{task}(\text{BigData, Alice, N_1})$.

For this to be well-defined, we require that each candidate st tgd $\theta$ 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].
to $t$. In the absence of candidates that fully explain $t$, we might include $\theta_1$ 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 $\theta$, we call $k \in K_\theta$ a possible explanation for $t$ under $\theta$ if there is a homomorphism $h : context_\theta(k) \to J$ with $h(k) = t$. Let $E(t, \theta)$ be the set of all possible explanations for $t$ under $\theta$. We call a labeled null unique if it appears exactly once in $context_\theta(k)$. For $k \in E(t, \theta)$, we define $null(k)$ to be the number of unique nulls in $k$ divided by the arity of $k$. So $null(k) = 0$ if $k$ contains only constants or labeled nulls used at least twice. Otherwise, $null(k) > 0$. We say that $k$ explains $t$ to degree $1 - null(k)$, and define the auxiliary function $covers(\theta, t)$ for $t \in J$ based on the maximal degree to which $t$ is explained by any tuple:

$$
covers(\theta, t) = \begin{cases} 
\max_{k \in E(t, \theta)} (1 - null(k)) & E(t, \theta) \neq \emptyset \\
0 & E(t, \theta) = \emptyset
\end{cases} \quad (11)
$$

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

$$
explains(M, t) = \max_{\theta \in M} \text{covers}(\theta, t) \quad (12)
$$

Equation (12) can be used to divide $J$ into three parts (Figure 3) for a given $M$: those tuples fully (d) or partially (e) explained through tuples in (b), and those that cannot be explained by $M$ 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 $C$ of candidate st tgds

$$
\text{Find } \arg\min_{M \subseteq C} \sum_{t \in J} [(1 - \text{explains}(M, t))] + \sum_{t \in K_\theta \setminus J} \text{error}(M, t) + \text{size}(M) \quad (13)
$$

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 $M$ that minimizes the error in (c) and maximizes the degree to which tuples in (d) and (e) are explained.

If all candidates are full, this optimization coincides with the one in (5), and so is NP-hard as well (see Section 3.4). In Section 5, we provide an efficient approximation algorithm for finding a high quality solution $M$.

4.3 Example of Selection over ST TGDS

We extend the running example to illustrate objective (13). We use a reduced candidate set $C' = \{\theta_1, \theta_3\}$ (Figure 1(d)) and the data in Figure 1(b)-(c), but omit the leader relation. A universal solution $K_{\theta_1}$ for $I$ contains the task tuples (BigData, Bob, Null$_1$) and (ML, Alice, Null$_4$), while a solution $K_{\theta_3}$ contains the task tuples (BigData, Bob, Null$_3$) and (ML, Alice, Null$_1$) and the or tuples (Null$_2$, IBM) and (Null$_4$, SAP).

For $\theta_1$, creates(·) is 1 for tuple task(BigData, Bob, Null$_1$), and 0 for all other tuples, and covers(·) is $2/3$ for task(ML, Alice, 111) and 0 otherwise. This is because task(ML, Alice, Null$_2$) partially explains the latter via a homomorphism mapping Null$_2$ to 111. Similarly, for $\theta_3$, creates(·) is 1 for task(BigData, Bob, Null$_1$) and org(Null$_3$, IBM), but 0 for task(ML, Alice, Null$_4$) and org(Null$_1$, SAP), which partially explain task(ML, Alice, 111) and org(111, SAP) to degree $2/3$.

The minimal value for the objective is that of the empty mapping, that is, the complexity term fulfills its role of guarding against overfitting on too little data here. But we also see that $\{\theta_1\}$ is preferred over $\{\theta_3\}$, which in turn is preferred over $\{\theta_1, \theta_3\}$. The reason is that while $\theta_3$ covers more tuples than $\theta_1$, it also produces more errors and is larger. If we add data for at least five more projects $X$ of the same kind as the ML one, i.e., pairs of tuples Proj(X,N,1) and task(X,Alice,111), the preferred mapping is $\{\theta_3\}$, as the empty mapping cannot explain the new target tuples, $\theta_1$ explains each to degree $2/3$, and $\theta_3$ fully explains them (while no mapping introduces additional errors).

5 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 (13)) as a program in probabilistic soft logic (PSL) [14], and uses 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 $\theta$, in our case) while maintaining the convexity of inference, which makes it possible to encode and efficiently solve model selection problems such as the mapping selection problem.

5.1 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:

\[ w : \ b_1(X) \land \ldots \land b_n(X) \rightarrow h_1(X) \lor \ldots \lor h_m(X) \quad (14) \]

where \( X \) is a set of universally-quantified variables, the \( b_i(X) \) and \( h_j(X) \) are atoms over (subsets of) the variables in \( X \), 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-\( t \)-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.

\[
I(\ell_1 \land \ell_2) = \max\{0, I(\ell_1) + I(\ell_2) - 1\} \\
I(\ell_1 \lor \ell_2) = \min\{I(\ell_1) + I(\ell_2), 1\} \\
I(\neg \ell_1) = 1 - I(\ell_1)
\]

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

\[
d_r(I) = \max\{0, I(\text{body}) - I(\text{head})\} \quad (15)
\]

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:

\[
f(I) = \frac{1}{Z} \exp[-\sum_{r \in R} w_r(d_r(I))] \quad (16)
\]

where \( w_r \) is the weight of rule \( r \) and \( Z \) is a normalization constant. PSL inference finds \( \arg\max_I f(I) \), 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].

5.2 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(\( \cdot \)) and creates(\( \cdot \)) functions defined in Section 4, respectively, and one predicate \( \text{in} \) 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 \( C \) will introduce a constant for every tuple in \( K_C \cup J \) and for every candidate in \( C \). We use the logical variable \( F \) for st tgds, and \( T \) for tuples. The CMD program consists of the following rules:

\[
\text{size}(F) : \ \text{in}(F) \rightarrow \bot \quad (17) \\
1 \rightarrow J(T) \rightarrow \exists F. \ \text{covers}(F, T) \land \text{in}(F) \quad (18) \\
1 \rightarrow \text{in}(F) \land \text{creates}(F, T) \rightarrow J(T) \quad (19)
\]

Rule (17) 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 \( \text{size}(f) \cdot (I(\text{in}(f)) - 0) \), and thus minimal if \( \text{in}(f) \) is false. Rule (18) states that if a tuple is in \( J \), there should be an st tgd in the set that covers that tuple, thus implementing the \( \text{creates}(\cdot) \) 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 (19) 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 \( K_C - J \)), no st tgd in the selected set should create it. This implements the \( \text{error}(\cdot) \) penalty. The advantage of this approach is that it reasons about the interactions between tuples and st tgds in a fine-grained manner. In Section 5.4 we show the rules combine to implement the mapping selection objective (13).

5.3 Prioritized Disjunction Rules

In first-order logic (with finite domains), formulas with existential quantifiers, such as Rule (18) 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. 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 (18), over st tgds to be selected). A prioritized disjunction rule is a rule:

\[
w : \ b(X) \rightarrow \exists Y. \ h_o(Y, X) \land h_i(Y) \quad (20)
\]

where \( b(X) \) is a conjunction of atoms, \( h_o(Y, X) \) is an observed atom and \( h_i(Y) \) is an atom whose value will be inferred. In our case, see (18), \( b(X) \) is \( J(T) \), \( h_o(Y, X) \) is \( \text{covers}(F, T) \), and \( h_i(Y) \) is \( \text{in}(F) \). The observed truth values of the \( h_o(Y, X) \) 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 \( h_o(Y, X) \). To efficiently support this comparison of alternatives, we introduce a \( k \)-prioritization for some natural number \( k \), restricting the truth values of \( h_o(Y, X) \) to \( \{0, k, \ldots, k/k\} \) 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 \( h_o(Y, X) \land h_i(Y) \) is either false or equivalent to \( h_i(Y) \). Since \( h_o(Y, X) \) is observed, for every grounding \( y \) of \( Y \), we can either drop the entire disjunct if \( h_o(y, X) \) is false or drop \( h_o(y, X) \) 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 \( j = 1, \ldots, k \) of the following form:

\[
\frac{w}{k} : b(x) \rightarrow \bigvee_{h_o(x,y) \in \{k/k\}} h_i(y)
\]

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

\[
\frac{w}{k} : b(x) \rightarrow \bigvee_{h_o(x,y) \in \{k/k\}} h_i(y)
\]

... and any \( m/k \) is the highest value \( h_o(x,y) \) takes for this \( x \) and any \( y \) with \( h_i(y) = 1 \), i.e.,

\[
m/k = \max_{\{y|h_i(y)=1\}} h_o(x,y)
\]

Then, the rules for \( j = 1, \ldots, m \) are satisfied (because their head evaluates to 1), and the ones for \( j = (m + 1), \ldots, k \) are not satisfied (because their head evaluates to 0). More precisely, their distance to satisfaction is the truth value of \( h_o(x,y) \), and let \( m/k \) be the highest value \( h_o(x,y) \) takes for this \( x \) and any \( y \) with \( h_i(y) = 1 \), i.e.,

\[
m/k = \max_{\{y|h_i(y)=1\}} h_o(x,y)
\]

We get a grounding of Rule (18) for every \( \theta \) in \( C \):

\[
1 : \top \rightarrow \exists F. \text{covers}(F,t) \land \text{in}(F)
\]

To complete the grounding, we apply PSL’s prioritized disjunction rules. Recall (cf. Section 4.2) that the \( \text{covers}(-) \) 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 \( \text{covers}(F,t) \) where \( k \) is the arity of the tuple \( t \). Thus we get for each \( t \in J \) a set of \( k \) ground rules, the \( i \)th of which is

\[
1/k : \top \rightarrow \exists \theta \in C, \text{covers}(\theta,t) \geq i/k
\]

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

\[
1 - \sum_{\theta \in \mathcal{M}} \max \text{covers}(\theta,t)
\]

due to prioritized disjunction rules rewriting, which equals (21). Thus, the CMD program optimizes Equation (13).
5.5 Collective Mapping Discovery

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

1) Compute truth values of evidence from \((I, J)\) and \(C\).
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 \(K_C\), and computes the truth values of the \(|C| \times |K_C - J|\) many creates(\(\cdot\)) atoms (based on Equation (9)) and of the \(|C| \times |J|\) many covers(\(\cdot\)) atoms (based on Equation (11)). 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.

6 Evaluation

We experimentally evaluate CMD on a variety of scenarios, both synthetic and real world, 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 both metadata and data. We also demonstrate that our prioritized disjunction rules enable efficient inference for complex mapping scenarios. We ran our experiments on an Intel Xeon with 24 x 2.67GHz CPU and 128GB RAM. Our implementation of CMD and instructions for reproducing all experiments can be found online.

6.1 Scenario Generation

Each of our scenarios consists of a data example \((I, J)\) for a pair of schemas \(S\) and \(T\) and a set \(C\) of candidate st tgd, which form the input for CMD, and a gold standard mapping \(M_G\) used to assess the quality of the solution. Scenario generation uses the following steps:

1) We generate schemas \(S\) and \(T\), correspondences, the initial data example \((I, J_G)\), and a gold standard mapping \(M_G\) that is valid and fully explaining for \((I, J_G)\) using the metadata generator iBench [18] and data generator ToxGene [29].
2) To create dirty metadata, we generate additional foreign key constraints and correspondences.
3) We use the implementation of the Clio [3] algorithm provided by ++Spicy [24] to generate the set of candidates \(C\) from the schemas, foreign key constraints and correspondences generated in previous steps, that is, based on metadata only.
4) We generate \(J\) starting from \(J_G\), introducing errors and unexplained tuples with respect to \(M_G\).

The rest of this subsection provides more details on this process, and Table 1 lists the parameters controlling it. All experimental parameters are for scenario generation; we set their defaults and ranges to produce realistic scenarios.

**STEP 1.** iBench uses transformation primitives to create realistic complex mapping scenarios. We chose a representative set of seven primitives.4 One invocation of this set creates a total of eight source and ten target relations, and seven st tgd in \(M_G\). Three of those are full, the other four all contain existentials used once or twice and include existentials that are shared between relations. To create larger scenarios, we invoke the set \(\pi_{\text{Invocations}}\) times. We set the size of \(I\) to \(\pi_{\text{TuplesPerTable}}\) per relation.

**STEP 2.** To obtain candidate st tgd with wrong join paths, we use iBench to add randomly created foreign keys to \(\pi_{\text{FKPerc}}\) % of the target relations. To obtain candidate st tgd making wrong connections between the schemas, we introduce additional correspondences as follows. We randomly select \(\pi_{\text{Corresp}}\) % of the target relations. For every selected target relation \(T\), we randomly select a source relation \(S\) from those of the iBench primitive invocations not involving \(T\) (so Clio [23] can generate \(M_G\) as part of \(C\)). For each attribute of \(T\), we introduce a correspondence to a randomly selected attribute of \(S\).

**STEP 4.** We restrict data instance modifications to errors and explainable tuples with respect to \(M_G\), as unexplainable tuples can be removed prior to optimization (cf. Section 3.3). In our scenarios, \(M_G \subseteq C\), and thus \(K_G \subseteq K_C\). So each tuple in \(K_C\) is either generated by both \(M_G\) and \(C - M_G\), only by \(M_G\) (i.e., an error tuple if deleted from \(J\)), or only by \(C - M_G\) (i.e., a new explainable tuple if added to \(J\)). As tuples in \(K_C\) may have nulls, we take into account homomorphisms when determining which of these cases applies to a given tuple. We randomly select a set \(J_{\text{New}}\) containing \(\pi_{\text{Explainable}}\) % of the potential new explainable tuples, and a set \(J_{\text{Err}}\) containing \(\pi_{\text{Errors}}\) % of the potential error tuples, and set \(J = J_G \cup J_{\text{New}} \setminus J_{\text{Err}}\).

**6.2 Evaluation of Solution Quality**

We assess quality by comparing the mapping \(M \subseteq C\) selected by CMD to (1) the correct mapping \(M_G\) produced by iBench and (2) the set \(C\) of all candidates produced by Clio, which serves as our metadata-only baseline. Since Clio does not use data, we are not confounding in our experiments how CMD uses data with other proposed approaches. 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 [30].

4. CP copies a source relation to the target, changing its name. ADD copies a source relation and adds attributes; DL does the same, but removes attributes instead; and ADL adds and removes attributes to the same relation. The number that are added or removed are controlled by range parameters, which we set to (2,4). ME copies two relations, after joining them, to form a target relation. VP copies a source relation to form two, joined, target relations. VNM is the same as VP but introduces an additional target relation to form a N-to-M relationship between the other target relations.
We use the core data exchange algorithm of +=Spicy [24] to obtain $K_M$ and $K_C$. The gold standard instance $K_G$ for $M_G$ is the original target instance $I$ 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. The distance between mappings is then defined as one minus the harmonic mean of these four measures; for full details, see Mecca et al [31]. We directly use the harmonic mean, which we call IQ-score $(K_1, K_2) \in [0, 1]$, where higher is better.

### 6.3 CMD Accuracy over Ambiguous Metadata

We begin by assessing the ability of CMD to handle ambiguous or dirty metadata and still identify a good mapping from the set of candidates. We increase the number of candidate st tgds by increasing the $\pi_{FKPerc}$ parameter from 0 to 10 percent and the $\pi_{Corresp}$ parameter from 0 to 100 percent. We use five scenarios per parameter setting, with an average of 800 source and 1000 target tuples. CMD always found the correct mapping, i.e., it resolved all metadata ambiguities based on the data example. In contrast, for Clio, which uses metadata only, the IQ-scores decreased with more imprecise evidence, as shown in Figure 4.

### 6.4 CMD Accuracy over Dirty Data

Our second experiment investigates the effect of imperfect data on mapping quality. We vary the percentage $\pi_{Errors}$ of added errors from 0 to 30% in steps of five, and the percentage $\pi_{Explainable}$ of additional explainable tuples from 0 to 100% in steps of 25. We set $\pi_{Corresp} = 100\%$ to maximize the number of potential explainable but undesired tuples. We consider five scenarios in each case, with 800 source tuples and 1000 tuples in the initial target instance $J$. The numbers of additional tuples obtained range from zero to 300 for errors and from zero to 1800 for explainable tuples.

In Figure 5, we plot IQ-score as we vary $\pi_{Errors}$ and $\pi_{Explainable}$. Generally, as the number of errors increases, i.e., more correct tuples are missing from the target instance, the quality of the mapping selected by CMD decreases, as there is less incentive to include candidates that would correctly explain such tuples, which results in lower IQ-score due to lower recall. Adding explainable tuples also generally decreases the quality of the mapping, as they provide incentives to include additional st tgds that, while explaining those dirty tuples, generally decrease precision and thus IQ-score. However, in the presence of significant numbers of errors, explainable tuples increase mapping quality. This happens whenever explainable tuples cause CMD to select st tgds in $C - M_G$ that are similar to $M_G$, e.g., omitting a target join on an existential variable, and when selecting those st tgds is preferred over the empty mapping. For scenarios with over one quarter additional explainable tuples, and even in the presence of a few (less than 10%) errors, CMD routinely finds mappings with high IQ-scores. This confirms that the fine-grained optimization score handles increasing noise levels gracefully.

### 6.5 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 tdg in $C$) is slow, taking up to 150 minutes in our largest scenarios; it would be easy to optimize this time, however that is not a focus of our current work.

**DATA SIZE.** 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 the average optimization time in CMD and average time to generate $C$ (the Clio baseline). CMD optimization times are comparable to Clio times even though we optimize over relatively large (3600 tuple) data examples.

**SCHEMA SIZE.** 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 relations total. The largest scenario involved 150 candidate st tgds, or over $10^{45}$ 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 (Figure 6(b)). Again, CMD optimization times are comparable to those of Clio, but for increasing schema size, the latter increases more quickly.
Fig. 6: Optimization time of CMD w.r.t. (a) size of data example, (b) schema sizes, and (c) maximal number of candidates explaining the same tuple. In (a) and (b), the dashed line is for the Clio baseline. In (c), \( \pi_{\text{SchemaSize}} \) is 20 (top), 10 (middle) or 5 (bottom).

**Complexity.** We vary the maximal number \( \pi_{\text{PDSize}} \) of candidate st tgds that explain each target tuple, which corresponds to the number of head atoms in prioritized disjunction rules. This is the main parameter determining the complexity of optimization. Our custom-made scenarios use a single primitive, and their complexity is controlled through two parameters, the arity \( \pi_{\text{Arity}} \) and the schema size \( \pi_{\text{SchemaSize}} \). The source has \( \pi_{\text{SchemaSize}} \) relations of arity \( \pi_{\text{Arity}} \), the target \( \pi_{\text{SchemaSize}} \cdot \pi_{\text{Arity}} \) relations of arity at most \( \pi_{\text{Arity}} \). The gold standard mapping has \( \pi_{\text{SchemaSize}} \) st tgds, and the number of potential candidates increases quadratically with both \( \pi_{\text{SchemaSize}} \) and \( \pi_{\text{Arity}} \). We set \( \pi_{\text{TuplesPerTable}} = 25 \).

We consider all combinations of \( \pi_{\text{SchemaSize}} \in \{5, 10, 20\} \) and \( \pi_{\text{Arity}} \in \{5, 10, 20\} \), and vary \( \pi_{\text{PDSize}} \) from 10 to 70 in steps of 20. We use one scenario for each combination. In Figure 6(c), we plot the optimization time for each value of \( \pi_{\text{SchemaSize}} \), aggregating over \( \pi_{\text{Arity}} \). In all scenarios, the mapping selected by CMD has perfect IQ-score. This result shows that optimization with prioritized disjunction rules is efficient even with 70 candidates explaining the same tuples, an order of magnitude higher than seen in our other tests.

### 6.6 CMD on Real Metadata and Data

The previous results show the power of CMD on benchmark datasets. Next we consider several real world scenarios.

**Amalgam.** We first consider the well-known Amalgam benchmark [32], using schema A1 as source and A3 as target. To construct a data example, we select a small subset of the data in A1. We use ideal correspondences, so this problem tests whether CMD selects st tgds with correct joins from the candidates generated by Clio. The final evaluation contains 18 relations and 2,502 tuples. For this scenario, CMD achieves IQ-score .99 and optimization time was under a minute. **Neuroscience.** We map Allen Brain Atlas (ABA), Kyoto Encyclopedia of Genes and Genomes (KEGG), Pharmacogenomics Knowledgebase (PharmGKB), and UniProt (Universal Protein resource) schemas to the Semantic MediaWiki Linked Data Environment (SMW-LDE) Ontology [33], [34]. ABA has one relation and 15 tuples; KEGG has four relations and 56 tuples; PharmGKB has four relations and 142 tuples; and UniProt has one relation and 15 tuples. The common target schema has 31 relations and 54 foreign keys. As with Amalgam, we construct data examples from the source instances and we use ideal correspondences. The CMD mappings for ABA, PharmGKB and UniProt achieved perfect IQ-scores. For KEGG, CMD got a score of .93.

CMD achieved a lower score on KEGG because it selected some candidates that reused labeled nulls for some attributes where the gold standard exchanged variables from the source. Our current scoring function cannot distinguish these, but could easily be adapted to do so.

### 7 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 [26] and ++Spicy [24] have extended this to richer forms of metadata (including equality-generating-dependencies). In addition, the role of data in resolving ambiguity or incompleteness in the metadata evidence has long been recognized, both in matching [35] and in schema mapping, where the Data Viewer [21] 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 [36]. 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 [22].

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 [37]. Finally, Sarma et al. [38] consider how to learn views (or full GAV mappings) 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. 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 [39] 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. Other work finds the top-k best matchings [40], while CMD finds only the best mapping. It is an open problem how to extend our optimization to top-k mappings.

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 [41], or to select join associations in the context of keyword-search based data integration [42]. 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 [43] 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. [43] 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 [44] for ontology alignment [45] and ontological mapping construction [46]. 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].

**8 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 has been supported by the Research Foundation Flanders (FWO). RJM is supported by NSF ERC, and LG by the National Science Foundation [IIS1218488]. The authors thank Boris Glavic, Patricia Arocena, Gianni Mecca, Donatello Santoro, and Radu Ciucanu for valuable help with iBench and +Spicy; and Craig Knoblock for the Neuroscience problem set.

**References**


