ABSTRACT

Title of dissertation: COLLECTIVE RELATIONAL
DATA INTEGRATION WITH
DIVERSE AND NOISY EVIDENCE

Alex Memory
Doctor of Philosophy, 2019

Dissertation directed by: Lise Getoor
Department of Computer Science

Driven by the growth of the Internet, online applications, and data sharing initiatives, available structured data sources are now vast in number. There is a growing need to integrate these structured sources to support a variety data science tasks, including predictive analysis, data mining, improving search results, and generating recommendations. A particularly important integration challenge is dealing with the heterogeneous structures of relational data sources. In addition to the large number of sources, the difficulty also lies in the growing complexity of sources, and in the noise and ambiguity present in real-world sources. Existing automated integration approaches handle the number and complexity of sources, but nearly all are too brittle to handle noise and ambiguity. Corresponding progress has been made in probabilistic learning approaches to handle noise and ambiguity in inputs, but until recently those technologies have not scaled to the size and complexity of relational data integration problems. My dissertation addresses key challenges arising from this gap in existing approaches.

I begin the dissertation by introducing a common probabilistic framework for reasoning about both metadata and data in integration problems. I demonstrate that this approach allows us to mitigate noise in metadata. The type of transformation I generate is particularly rich – taking into account multi-relational structure in both the source and target databases. I introduce a new objective for selecting this type of relational transformation and demonstrate its effectiveness on particularly challenging problems in which only partial outputs to the target are possible. Next, I present a novel method for reasoning about ambiguity in integration problems and show it handles complex schemas with many alternative transformations. To discover transformations beyond those derivable from explicit source and target metadata, I introduce an iterative mapping search framework. In a complementary approach, I introduce a framework for reasoning jointly over both transformations and underlying semantic attribute matches, which are allowed to have uncertainty. Finally, I consider an important case in which multiple sources need to be fused but traditional transformations aren't sufficient. I demonstrate that we can learn statistical transformations for an important practical application with the multiple sources problem.

COLLECTIVE RELATIONAL DATA INTEGRATION WITH DIVERSE AND NOISY EVIDENCE

by

Alex Memory

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Professor Lise Getoor, University of California Santa Cruz (Advisor, Co-chair)
Professor Dana Nau, University of Maryland (Co-chair)
Professor Héctor Corrada Bravo, University of Maryland
Professor Louiqa Raschid, University of Maryland
Professor Alan Ritter, The Ohio State University

© Copyright by
Alex Memory
2019

Acknowledgments

This work was possible because of help from many others. First, I thank Lise Getoor for her endless insights and encouragement. Credit also goes to our collaborators Angelika Kimmig and Renée J. Miller, whose brilliance and knowledge helped overcome many challenges along the way. I also thank my committee: Dana Nau, Héctor Corrada Bravo, Louiqa Raschid, and Alan Ritter.

Fellow LINQS members, including Arti Ramesh, Ben London, Bert Huang, Dhanya Sridhar, Eriq Augustine, Hui Miao, James Foulds, Jay Pujara, Shobeir Fakhraei, Stephen Bach, and Theodoros Rekatsinas were a wonderful source of help and ideas. Thanks to Boris Glavic, Gianni Mecca, and Radu Ciucanu for generously sharing their work and to Amol Deshpande, Graham Mueller, Henry Goldberg, Leora Morgenstern, Rafael Alonso, and Ted Senator for many helpful discussions. I especially thank my family and friends for their patience and encouragement.

Some work was done as part of the ELLIPSE project, which was supported by the Office of the Director of National Intelligence (ODNI) and the Intelligence Advanced Research Projects Activity (IARPA) via the Air Force Research Laboratory (AFRL) contract number FA8750-16-C-0114. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, AFRL, or the U.S. Government.

Table of Contents

List of Figures

Chapter 1: Introduction

Driven by the growth of the Internet, online applications and data sharing initiatives, available structured data sources are now vast in number. Publicly available data sources alone now number in the millions [1]. There is a growing need to integrate these structured sources to support a huge variety data science tasks, including predictive analysis, data mining, improving search results, and generating recommendations [2]. However, the problem of integrating structured data is far from solved.

A particularly important integration challenge is dealing with the heterogeneous structures of sources. Due to the flexibility of data representations, and varying needs of data creators, even sources containing closely related data are often structured very differently. Critical data science tasks, and related integration tasks such as entity resolution, can't begin until we first overcome this challenge of structural heterogeneity.

While integrating sources with well-understood structures is itself an area of research, it is now a challenge just to understand the structures of the vast number of sources – both individual sources and their versions over time [3]. Part of the difficulty also lies in the growing complexity of sources, with some schemas having

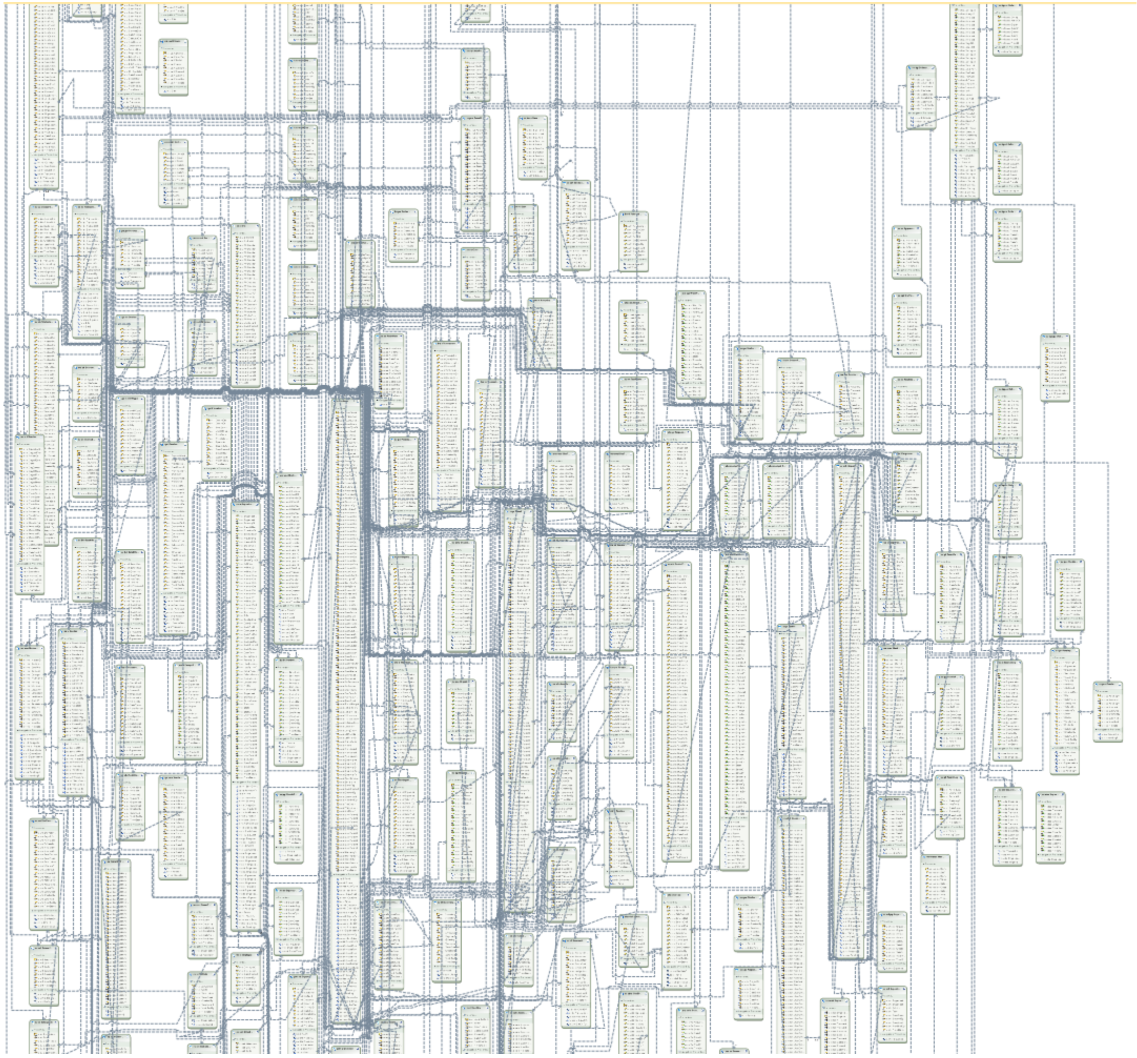


Figure 1.1: An example of a complex database schema.

hundreds of thousands of concepts [4]. Figure 1.1 shows an example of a complex schema.[1] It is critical to automate structured data integration as much as possible to overcome these difficulties.

Much progress has been made to develop automated integration approaches for handling the number and complexity of sources. However, nearly all are based on brittle technologies that cannot also handle noise and ambiguity in metadata and data. Corresponding progress has been made in probabilistic learning approaches to

[1] *GlobalServices* model: https://mattduffield.files.wordpress.com/2011/06/large-model.png

Source

A	B	C
a	b	c
a'	b'	c'

D	E
d	e
d'	e'
d''	e''

Target

F	G
f	g
f'	g'

H	I	J
h	i	j
h'	i'	j'

Figure 1.2: A notional data source (left) and target database (right).

handle noise and ambiguity in inputs to prediction and mining tasks. However, until recently these technologies have not scaled to the high size and complexity needed for structured data integration.

In the following section, I briefly summarize seven key challenges for structured data integration arising from this gap in existing approaches, and my contributions helping to address each.

1.1 Challenges

I focus specifically on *relational* data integration; relational data is very widely used, and is the data representation used by most database management and data analysis systems. As illustrated in Figure 1.2, inputs to a relational data integration problem include a database with source data (left) and a *target* database (right).

In the notional example, there are two source relations and two target relations, a total of ten attributes $(A, B, \ldots, J)$, and a total of nine tuples. Our over-all goal is to transform data in tuples of the source so it can be used in the target schema, and any applications using that schema.

Substantial work has focused on basic parts of this problem, such as transforming a single attribute at a time, transforming a single source relation at a time, or filling a single target relation at a time. I focus on the more general integration setting with multiple source relations and target relations. All integration tasks include choices incorporating elements of the source and target; in this more general setting, we add additional complexity from integration choices within the source and within the target, e.g., joining relations on some attributes. This makes the space of possible transformations very large.

I will now briefly describe seven key challenges in relational data integration, and my contributions to address each.

Noisy Metadata. An important integration technique is to use metadata accompanying a source, along with metadata for the target, to derive possible transformations from the source to the target. Metadata includes names of relations, names of attributes, and schema constraints. For example, foreign key (FK) constraints are a kind of metadata that is often used to derive possible join paths within schemas [5]. In Figure 1.3, I illustrate one notional FK that is correct (a). However, in practice FKs can be noisy, e.g., using the one marked (b) could create incorrect tuples in the target. An important insight is that *combining metadata with data* could mitigate

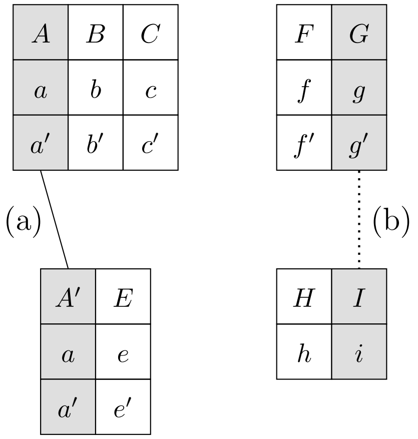


Figure 1.3: Noisy metadata example.

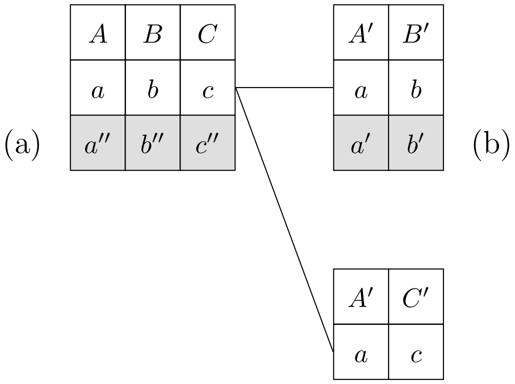


Figure 1.4: Noisy data example.

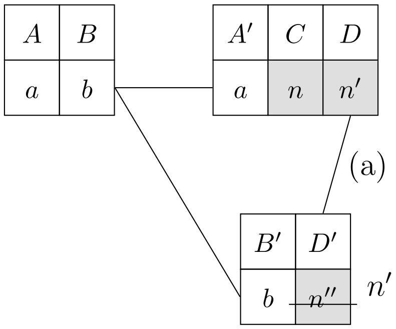


Figure 1.5: Partial outputs example.

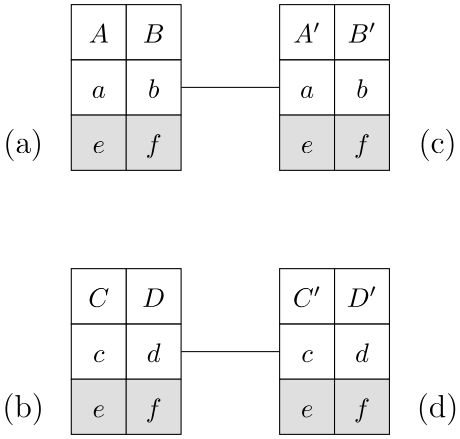


Figure 1.6: Ambiguous data example.

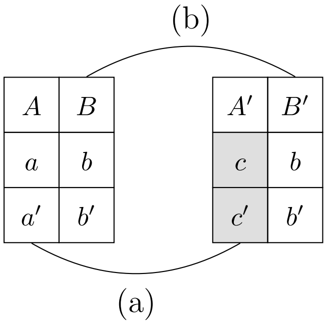


Figure 1.7: Uncertain semantics example.

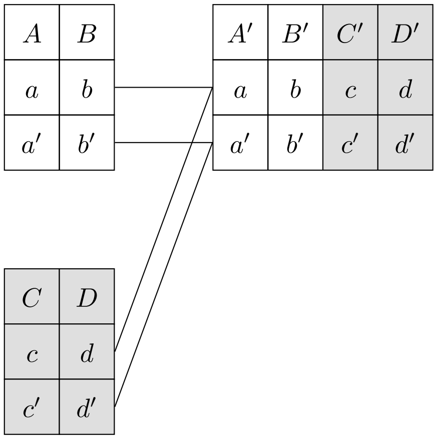


Figure 1.8: Multiple sources example.

metadata noise. I will show how, using examples of the same data appearing in both a source and the target, we can overcome several important types of noise in metadata.

Noisy Data. In Figure 1.4, I show an example of data appearing in both the source and target: tuple (a, b, c) in the source is partitioned vertically into (a, b) and (a, c) in the target. Unfortunately, in practice data examples can contain noise as well: the tuple marked (a) has no corresponding tuples in the target, and the reverse is true for (b), reducing our confidence in the correct transformation. Earlier approaches to use data this way have been brittle, generating incorrect transformations, or none at all, when exposed to noise. I will show that, with realistic levels of data noise, a flexible *probabilistic* framework can handle data noise and produce correct transformations.

Partial Outputs. Sources sometimes lack data to fill all attributes in the target. As illustrated in Figure 1.5, when transforming source tuple (a, b), we only have values to fill target attributes A' and B', which are now vertically partitioned into separate target relations. With existing approaches, transformations would fill the three empty output attributes C, D, and D' with null values. By default, nulls are unique, e.g., n, n', and n''. This works for attribute C, but placing unique nulls in A' and B' loses the relationship between a and b. Foreign key (a) links attributes D and D', suggesting a way to preserve the relationship between a and b. To do this, we need a transformation that creates a shared null, e.g., putting n' in both D and D'. However, existing relational learning approaches do not consider nulls shared across target relations, as they add significant complexity. We need a new kind of relational learning objective that takes multi-relational target structure into account. I will introduce a new *homomorphism-based objective* to do this, and show that it scales well to realistically large and complex integration problems.

Ambiguous Data. Even in ideal example data, there can still be ambiguity due to the nature of the data. In Figure 1.6, tuples (a, b) and (c, d) are unambiguous, suggesting one transformation of (A, B) to (A', B'), and a second from (C, D) to (C', D'). However, tuple (e, f) appears in all four relations, leading to ambiguity whether to pair tuples marked (a) with (c), or (a) with (d), (b) with (c), etc. We are left unsure which transformations are correct. In actual problems with ambiguous data, one strategy could be to consider all transformations and all tuples holistically to see if the aggregate evidence prefers some alternatives over others. Existing approaches have typically not attempted this type of analysis, as it has the potential to add combinatorial complexity to the problem. I will show that reasoning both probabilistically and *collectively* over alternate transformations is indeed possible. Specifically, I will show that a new type of probabilistic dependency called prioritized disjunction rules (PD) that we develop for this purpose scales well to highly complex integration problems with many alternatives.

Flawed Transformations. I have described how we can derive possible transformations from metadata, and combine them with data in a probabilistic, collective framework to handle noise, ambiguity and partial outputs. However, ideal transformations may not be derivable from metadata because parts of metadata are missing or too noisy to be recoverable. For example, if in Figure 1.3 the FK marked (a) were missing, we would not derive the necessary join over those relations using conventional techniques. It is possible to systematically tweak parts of transformations, then test which variant fits the data best. Unfortunately, an undirected search over variants is impractical – we need guidance to narrow the search. I will show that our collective, probabilistic framework using metadata and data provides an efficient source of guidance when integrated into a *boosted search* over transformations.

Uncertain Semantics. In addition to the metadata within each schema, sometimes an additional form of metadata called attribute *correspondences* are available that crosses schemas. Correspondences are links from attributes in the target schema to attributes in the source with the same meaning. For example, if an attribute in the target contains names of people, correspondences would identify attribute(s) in the source that also contain names of people. Correspondences are helpful for deriving possible transformations, and a significant body of past work focuses solely on the problem of finding correspondences. Unfortunately, correspondences can be noisy, and have varying levels of confidence, which most existing transformation approaches are too brittle to handle. In Figure 1.7, I illustrate one notional correct correspondence (b) and one incorrect correspondence (a). I consider the case that some correspondences can be found *jointly* with transformations as a strategy to improve and harmonize both. I propose and evaluate a joint probabilistic transformation framework that finds missing correspondences and handles their inherent noise and uncertainty in a principled way.

Multiple Sources. I have described approaches to overcome noise and ambiguity to reveal the true source structure to transfer to the target. However, in some important cases the source structure is inherently lacking, and we need to synthesize missing connections between source relations to provide data fitting the target schema. This situation arises frequently when multiple original sources are ingested into a common repository – sometimes referred to as a *data lake*. In Figure 1.8, I illustrate an example in which data from two source relations must be merged vertically – not by union. By themselves, the source relations provide no way to discover that target tuple (a, b, c, d) is correct, instead of (a, b, c', d') or (a', b', c, d). We lack the explicit data and metadata we would need to make a conventional transformation, but I will show how – with the addition of some background knowledge – we can learn a *statistical transformation* for this source fusion problem.

1.2 Structure of Dissertation

My contributions to address the challenges listed in Section 1.1 are based on foundational work in data integration and structured prediction. In Chapter 2, I provide brief introductions of past work in these two areas. Specifically, most of the work presented here leverages two powerful and complementary technologies: (a) from the data integration literature, I adopt a particularly rich form of transformation known as source-to-target tuple generating dependencies (st tgd), which allow us to represent transformations with multiple source relations, multiple target relations, and partial outputs; and (b) from the structured prediction literature, I use the recently developed probabilistic soft logic (PSL) [6] language, serving as a foundation to encode a wide variety of integration objectives and handle many forms of noise and ambiguity.

Central to several integration challenges is a framework for reasoning about both metadata and data. In Chapter 3, I introduce a probabilistic framework based on PSL to do this, using a novel objective for selecting st tgd-based mappings from noisy metadata and data in Section 3.3 that optimizes both coverage and correctness. In Section 3.4, I extend the objective using homomorphic equivalence to take into account partial outputs, significantly expanding the practical applications of the integration approach.

In Chapter 4, I present and evaluate a novel extension to the PSL language called prioritized disjunction rules. This new kind of rule enables an important new kind of probabilistic dependency for reasoning about ambiguity over large numbers of alternative transformations. I use this new kind of rule in Chapters 3, 5, and 6. Material from Chapters 3 and 4 were published in [7], [8], and [9].

In Chapter 5, I build on foundations from Chapters 3 and 4, creating an iterative search framework to address flaws in mappings. Specifically, I leverage the PSL-based learning objective from earlier chapters in a boosted search algorithm to focus the search towards the ideal transformation. I show that the search algorithm corrects several important classes of flaws in st tgd-based transformations. The material in Chapter 5 is from work that is in progress.

In Chapter 6, I introduce a complementary framework to the ones presented in Chapters 3 and 5, which reasons jointly over both correspondences and st tgd-based transformations. I evaluate the framework and show that it is able to find correct correspondences and transformations, even when starting from just the source and target schemas.

In Chapter 7, I consider the important multiple sources problem, which requires inferences that are impossible with pure st tgd-based transformations. In this work, I learn statistical transformations using PSL rules, background knowledge, and external similarity measures. I then evaluate it on an important practical application. Material from Chapter 7 was published in [10].

In Chapter 8, I conclude with a summary of my contributions and potentially fruitful new directions for future research based on this work.

1.3 Summary of Contributions

In this dissertation, I present novel approaches to seven key challenges of relational data integration. Common to all contributions is the combination of state-of-the-art advances in data integration and structured prediction in order to incorporate diverse inputs and to handle noise. In summary:

1. **Combining metadata and data:** Metadata is useful for deriving possible transformations, but breaks down as a single approach in realistic settings because of noise. To overcome the huge space of possible corrections to the noise, I introduce a framework combining metadata with data. In an extensive empirical evaluation, I test this ability on a wide variety of metadata structures and varying metadata noise, demonstrating the power of combining data with metadata.

2. **Probabilistic framework:** Simply using patterns present in data will produce bad transformations, because of noise present in data as well. Instead of assuming perfect example data, I use a probabilistic approach that accommodates noise. I evaluate this ability on a wide variety of data noise, demonstrating the ability to handle significant levels of noise.

3. **Homomorphism-based objective:** A correct transformation transfers as much data as possible from a source, and also provides as much information as possible about remaining unknowns via shared labeling on nulls across multiple relations. To do this, I overcome the common restriction of learning a single relation at a time by introducing a new learning objective based on multi-relation homomorphisms.

4. **Collective, prioritized disjunction rules:** Inherent ambiguity is possible even in ideal integration inputs, resulting in a potentially intractable number of alternative transformations. In several empirical evaluations, I demonstrate that the new prioritized disjunction rules are efficient and handle highly complex sets of alternate transformations. In problems combining noisy metadata, noisy data, partial outputs, and ambiguous data, those challenges can interact to create even more complexity so I include combinations of those challenges in evaluation problems.

5. **Boosted search:** Many sources are available in such a raw form that deriving useful possible transformations is difficult. A naive search over the huge space of possible transformations would be intractable, but I demonstrate that our

collective, probabilistic framework is a good basis for guiding the search. I evaluate the search on a novel set of generated integration scenarios, as well as real data sets.

6. **Joint matching and mapping:** A huge body of work exists to discover correspondences, but most existing mapping approaches are highly sensitive to their uncertainty and noise. I propose a new technique for learning matches and mappings jointly and demonstrate its potential on several generated and real data sets.

7. **Statistical transformations for source fusion:** A growing need is to fuse multiple sources that have been ingested to large repositories, e.g., data lakes, yet conventional rule-based transformations aren't suitable for this problem. I introduce a new approach using statistical transformations and demonstrate its effectiveness on an important practical application.

Chapter 2: Related Work

I briefly review related work in data integration and structured prediction.

2.1 Data Integration

Data integration has the broad goal of providing a unified view over multiple data sources to applications and users. In this section, I review two areas of work in data integration that are particularly important to this dissertation: schema matching and mapping has the goal of discovering relationships between source and target schemas; and data exchange and metadata management has the goal of managing and applying those relationships in applications.

2.1.1 Schema Matching and Mapping

An important data integration task is finding semantically related attributes known as *schema matchings* or *correspondences* [11]. Much work on schema mapping has focused on this initial step, often combining multiple similarity measures to discover matches [12]. Approaches for matching also extend beyond 1-to-1 matches to include complex matches, e.g., Dhamankar et al. [13] A related problem is ontology alignment, which finds correspondences between classes of objects and between binary predicates, e.g., PARIS [14]. Hu et al. [15] align ontologies using first-order Horn rules.

Schema mapping discovery, the task of identifying complex structural relationships between relational data sources, has a long and rich tradition [16, 17, 18], yet it remains a challenging problem. Schema mappings are a central component of applications that integrate information from a variety of data sources using different vocabularies and structured representations. Approaches for discovering mappings build on techniques for query discovery [5], learning from examples [18, 19] and statistical relational learning [20]. The Clio project showed how constraints within schemas can be used to infer schema mappings [5, 21]. HepTox [22] and ++Spicy [23] have continued this line of research using richer forms of metadata (including equality-generating-dependencies) to guide mapping discovery. The ++Spicy system [24] also includes a verification step following matching and mapping to improve the quality of discovered st tgds based on data instances.

The role of data in resolving ambiguity or incompleteness in the metadata evidence has long been recognized, both in matching [25] and in schema mapping [26, 27]. The Data Viewer [26] and Muse [27] systems use source and target data interactively to help a user understand, refine or correct mappings suggested by a mapping design system. A complementary approach that uses data only is often called example-driven schema-mapping design [28]. This approach uses a set of data examples (a set of pairs of a source instance and a target instance) to determine if there exists a mapping (a set of st tgds) that fit the examples [18]. If such a mapping exists, then the system returns the most general such mapping. For many examples, there will be no fitting mapping and the system (Eirene) interactively guides a user in refining her data to identify tuples that are causing the failure [29]. Another approach to interactive mapping discovery from data casts the problem as an algorithmic learning problem, where a user is asked to label a series of examples as positive or negative in order to learn a mapping [30]. And of course there is work on learning views from data [31] where a view is a restricted form of st tgds where the target expression is a single relational atom with no existentials.

Gottlob and Senellart [32] define the schema-mapping discovery problem from a single data example (a source and target instance pair) as an optimization problem. Given a data example, that is, a source instance I and target instance J, the problem is to find a mapping M that is *valid* (meaning that $(I, J) \models M$) and that is *fully explaining* for (I, J) (meaning that every fact in J belongs to every target instance K such that (I, K) satisfies M). The optimization problem then is to search among all valid, fully-explaining mappings, to find the minimal cost mapping (where cost is defined precisely, but intuitively measures the size of the mapping). Even for learning a set of st tgds with no existential quantifiers, finding optimal mappings is shown to be DP-hard (both NP-hard and co NP-hard). Using this framework, ten Cate et al. [33] provide an approximation algorithm for finding near optimal schema mappings from a data example when mappings are limited to have a single atom (relation) in the head and body.

2.1.2 Data Exchange and Metadata Management

The rich fields of data exchange and metadata management provide a foundation to understand and use schema mappings. Kolaitis [34] gives an overview of the major concepts used in data exchange and metadata management and some of the most important recent advances in the field. Doan et al. [35], give a recent review of practices in the field and current challenges.

There is an increasing need for flexible mappings that can be reused and combined within the larger goal of metadata management. Arenas et al [36] introduce formal foundations to understand qualities of mappings needed for metadata management, including the ability of a mapping to transfer source information and the ability to reverse mappings. Fagin et al. [37] also consider reversing data exchange — that is, the ability to swap the source and target schemas in a mapping. In the process, they consider important properties of nulls in mappings which are broadly important whenever mappings are used in an application. Furthermore, Arocena et al. [38] consider the problem of inventing values for nulls during data exchange.

Interrelatedness between the mapping and data cleaning tasks are an area of recent progress in data exchange. In particular, Mecca et al. [39] introduce the *core* schema mapping, a formal description of data exchange solutions that minimize duplication which would otherwise need to be resolved with a separate cleaning step. More recently, LLUNATIC [40] puts in a common framework core data exchange solutions using mappings and also flexible cleaning dependencies that can further reduce errors and duplication.

Evaluating a schema mapping is challenging because mappings can differ, yet still be equally useful [41]. To avoid this problem, IQ [42] is a flexible evaluation measure for schema mappings that judges data exchange results rather than mappings themselves. To provide a benchmark set of diverse integration problems, Arocena et al. introduce iBench benchmark [43]. Finally, with recent advances, data exchange can be applied to even larger integration problems; Dong et al. [1] give an overview of unique problems that arise with such large scale data.

2.2 Structured Prediction

Structured prediction is a machine learning approach in which predictions are structured objects. A particularly flexible approach to structured prediction is to base it on probabilistic graphical models (PGM) [44]. PGMs provide ways to build statistical models for a variety of domains where there are detailed dependencies between random variables. In particular, PGMs give principled ways to handle missing information and to make predictions involving structured combinations of variables. Markov logic [45] and probabilistic soft logic [6] are examples of PGMs in which templated dependencies can be expressed using first-order logic, which makes specifying models especially easy. Both supervised and unsupervised learning are possible with PGMs. For example, without labeled training data, Poon et al. [46] train a Markov logic network that performs coreference resolution.

Structured prediction using PGMs has been used successfully for data integration. For example, Niepert et al. [47] use declarative rules and probabilistic inference using Markov logic to find matches between ontologies with binary relations on the source and target side. Additionally, PGMs have been used to clean data extracted from multiple unstructured data sources to form knowledge graphs [48].

Structural dependencies have also been studied in the inductive logic programming and relational learning (RL) communities [49]; the problem of learning mapping dependencies across relational schemas is closely related to the problem of learning dependencies in RL. Due to their complexity, the space of possible structured predictions is often extremely large or even infinite. For example, Lee et al. [50] use structured prediction to search a large space of possible solutions — up to a fixed time horizon — for a planning problem. Many approaches to structured prediction, including PSL, are closely related to constraint satisfaction. In early work, Beck et al. [51] use constraints to search efficiently a space of solutions with no fixed horizon, but the constraints do not define a probability distribution.

2.2.1 Probabilistic Soft Logic

In this section I briefly define the PSL language, as it is used throughout the rest of this dissertation. A PSL program is a set of weighted function-free rules with conjunctive bodies and disjunctive heads

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

where $\vec{X}$ is a set of universally-quantified variables, the $b_i(\vec{X})$ and $h_j(\vec{X})$ are atoms over (subsets of) the variables in $\vec{X}$, 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). Instead of talking about a ground rule being satisfied, PSL focuses on such 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). However, in order to achieve efficient inference, PSL uses *soft truth values* from the interval $[0, 1]$ instead of the usual Boolean ones. This requires suitable generalizations of the logical connectives as well, which is done using the *Lukasiewicz t-norm* and its corresponding *co-norm*. These relaxations are exact at the extremes, but provide a consistent mapping for values in-between. Given an interpretation I of all ground atoms constructed from the predicates and constants in the program, the formulas for the relaxation of the logical conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$) are as follows:

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

The distance to satisfaction of a ground rule $r = \text{body} \rightarrow \text{head}$ is then

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

For instance, consider the rule $w: \quad a \wedge b \rightarrow c \vee d$ with truth values $I(a) = 0.7$, $I(b) = 0.8$, $I(c) = 0.3$ and $I(d) = 0.1$. The truth value of $a \wedge b$ is $\max\{0, 0.7+0.8-1\} = 0.5$, that of $c \vee d$ is $\min\{0.3 + 0.1, 1\} = 0.4$, and the distance to satisfaction thus $0.5 - 0.4 = 0.1$.

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:

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

where w_r is the weight of the rule r and Z is a normalization constant. The inference task PSL solves is to find $\arg\max_I f(I)$, 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 *given* 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 can be converted to a traditional, Boolean-valued interpretation using a greedy rounding strategy. The quality of this solution is guaranteed to be at least $^3/_4$ that of the optimal discrete solution, with respect to the total weighted distance-to-satisfaction objective. Bach et al. [52] provide the full technical details of the results with rounding. Additionally, Bach et al. [53] showed that inference in PSL models can be one hundred times faster than inference in Markov logic, while still being more accurate.

Chapter 3: Mappings from Metadata and Data

3.1 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 [54]. 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 and structured prediction.

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 [5, 21]. Metadata in the form of query logs has been used to select mappings that are most consistent with frequently asked queries [55]. Many different approaches use data to refine a mapping or to select a mapping from among a set of schema mappings [27, 28, 32, 56, 24, 18, 33, 57]. 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 [19]. 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. [57] 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) [6], 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 [58] and data fusion [59, 60]. 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 tgds 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 tgds. 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 tgds. 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.

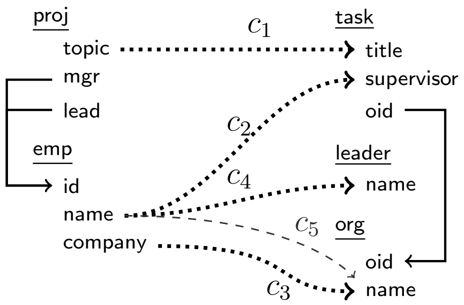
We perform an extensive empirical validation of our approach. We use the integration benchmark iBench [43] to test CMD on a wide variety and large number of mapping scenarios. We use IQ-Meter [61], a multi-criterion evaluation measure, to confirm the quality of CMD's output. We compare CMD with a baseline 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.

Section 3.2 illustrates the key challenges with an example. Section 3.3 introduces the selection problem for st tgds without existentially quantified variables, and Section 3.4 extends this to st tgds. Section 3.5 introduces our solution using PSL and our extension of PSL with *prioritized disjunctions*. We discuss experiments in Section 3.6 and related work in Section 3.7.

3.2 Motivating Example

Figure 3.1a 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 3.1b, 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 [26, 27], or in correcting a set of data examples so that a "best fitting" mapping exists [29]. 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



(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.

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

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

Figure 3.1: Motivating example; see Section 3.2 for details.

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 [62] or ++Spicy [23], or may have been mined from a query log [55].

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 3.1d, focusing first on our earlier data example (Figure 3.1b) 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 Alice is wrong and 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 3.1c, 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 3.1c 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

[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.

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 3.1a. If a matcher incorrectly suggested correspondence c_5, 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 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 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 [19].

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.

3.3 Mappings with Full Outputs

We first define mapping selection for full st tgds [54], that is, st tgds without existentially quantified variables, and extend our definitions to arbitrary st tgds in Section 3.4.

3.3.1 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 $\mathbf{T}$ [54]. 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* [63].

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 [62], HepTox [22], and ++Spicy [23] use schema constraints, while U-Map [55] 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.

3.3.2 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. [63], 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 $R(x_1, \ldots, x_n)$ in $\phi(x)$, $R(h(x_1), \ldots, h(x_n))$ 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. [63] 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 [32] 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 3.1d is not valid with respect to the data example in Figure 3.1c. However, if we add the tuple $t' = \mathsf{leader(Pat)}$ 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 [32], 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, else t is an *unexplained tuple*. As with validity, we would like to permit exceptions, that is, a few tuples in J that are unexplained, meaning J is not fully explained.

Example 2: Consider again θ_5 of Figure 3.1d. 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 $\mathsf{leader(Joe)}$, then θ_5 would still be valid, but $\mathsf{leader(Joe)}$ is an unexplained tuple.

3.3.3 Collective Selection over Full Mappings

We now define an optimization problem for finding a mapping $\mathcal{M} \subseteq \mathcal{C}$ 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 $\mathcal{M}$.

The first two are formalized through functions that, for a candidate set $\mathcal{M}$, compare the given target instance J to the solution for $\mathcal{M}$ and I, i.e., check how many tuples in J are unexplained (collectively) by $\mathcal{M}$, and how many tuples resulting from data exchange with each st tgd in $\mathcal{M}$ are not in J. Figure 3.2 illustrates these different kinds of tuples.

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 so canonical universal solutions are unique. We define $\mathsf{creates}_{\mathsf{full}}(\theta, t)$ as follows:

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

We then define $\mathsf{error}_{\mathsf{full}}(\mathcal{M}, t)$ for a tuple $t \in K_{\mathcal{C}} - J$ (Figure 3.2 left side) to be the number of st tgds in $\mathcal{M}$ for which t is an error.

$$\mathsf{error}_{\mathsf{full}}(\mathcal{M}, t) = \sum_{\theta \in \mathcal{M}} (\mathsf{creates}_{\mathsf{full}}(\theta, t)) \tag{3.1}$$

Correspondingly, for the tuples in J (Figure 3.2 right side), we define the function $\mathsf{explains}_{\mathsf{full}}(\mathcal{M}, t)$, which checks whether such a tuple is explained by $\mathcal{M}$.

$$\mathsf{explains}_{\mathsf{full}}(\mathcal{M}, t) = 1 \text{ if } t \in J \cap K_{\mathcal{M}} \text{ and } 0 \text{ otherwise} \tag{3.2}$$

We call tuples in $J - K_{\mathcal{C}}$ that cannot be explained by any st tgd in $\mathcal{C}$ *unexplainable tuples* (Figure 3.2(g)).

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

$$\mathsf{size}(\mathcal{M}) = \sum_{\theta \in \mathcal{M}} (\text{number atoms in } \theta) \tag{3.3}$$

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

$$\textbf{Find} \quad \begin{aligned} \arg\min_{\mathcal{M}\subseteq\mathcal{C}} (& \sum_{t\in J}[1 - \mathsf{explains}_{\mathsf{full}}(\mathcal{M}, t)] \\ & + \sum_{t\in K_{\mathcal{C}}-J}[\mathsf{error}_{\mathsf{full}}(\mathcal{M}, t)] \\ & + \mathsf{size}(\mathcal{M})) \end{aligned} \tag{3.4}$$

As we show in Section 3.3.4, 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 [32]. 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 3.2, our goal is to find an $\mathcal{M}$ that jointly minimizes the number of unexplained but explainable tuples (those in (f)), the number of errors (those in (c)) and the size of $\mathcal{M}$. Note that every $\mathcal{M} \subseteq \mathcal{C}$ receives a constant penalty

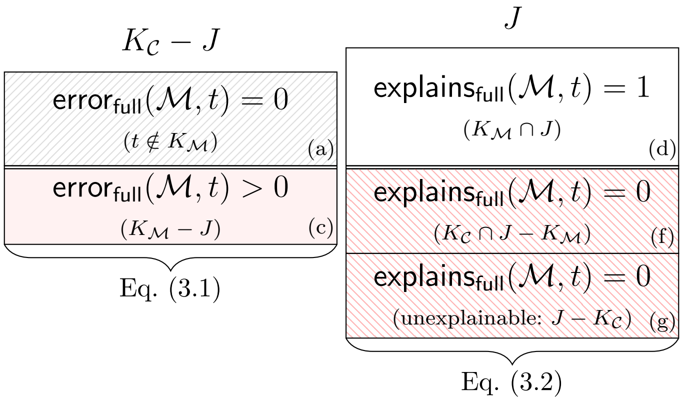

$K_{\mathcal{C}} - J$
J
$\mathsf{error}_{\mathsf{full}}(\mathcal{M}, t) = 0$
$(t \notin K_{\mathcal{M}})$
(a)
$\mathsf{error}_{\mathsf{full}}(\mathcal{M}, t) > 0$
$(K_{\mathcal{M}} - J)$
(c)
Eq. (3.1)
$\mathsf{explains}_{\mathsf{full}}(\mathcal{M}, t) = 1$
$(K_{\mathcal{M}} \cap J)$
(d)
$\mathsf{explains}_{\mathsf{full}}(\mathcal{M}, t) = 0$
$(K_{\mathcal{C}} \cap J - K_{\mathcal{M}})$
(f)
$\mathsf{explains}_{\mathsf{full}}(\mathcal{M}, t) = 0$
(unexplainable: $J - K_{\mathcal{C}}$)
(g)
Eq. (3.2)

Figure 3.2: Illustration of functions $\mathsf{error}_{\mathsf{full}}(\cdot)$ and $\mathsf{explains}_{\mathsf{full}}(\cdot)$.

for unexplainable 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 $\mathsf{error}_{\mathsf{full}}(\cdot)$ considers each candidate in $\mathcal{M}$ individually, and sums the number of errors made by each. That is, if two st tgds $\theta_i \in \mathcal{M}$ and $\theta_j \in \mathcal{M}$ 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.

3.3.4 Mapping Selection is NP-hard

The $\mathsf{error}_{\mathsf{full}}(\cdot)$ and $\mathsf{size}(\cdot)$ terms of (3.4) are modular and act as constraints on the supermodular $\mathsf{explains}_{\mathsf{full}}(\cdot)$ 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* (3.4) *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 $\mathbf{S}$, $\mathbf{T}$, a data example (I, J), a set $\mathcal{C}$ of candidate *full* st tgds, and a natural number m, is there a selection $\mathcal{M} \subseteq \mathcal{C}$ with $F(\mathcal{M}) \leq m$?

where $F(\mathcal{M})$ is the function minimized in (3.4), i.e.,

$$\begin{aligned} F(\mathcal{M}) = & \sum_{t \in J} [1 - \mathsf{explains}_{\mathsf{full}}(\mathcal{M}, t)] \\ & + \sum_{t \in K_{\mathcal{C}} - J} [\mathsf{error}_{\mathsf{full}}(\mathcal{M}, t)] + \mathsf{size}(\mathcal{M}) \end{aligned}$$

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 the parts of the instance as follows:

$$
\begin{aligned}
&\mathbf{S} = \{R_i/2 \mid R_i \in R\} \\
&\mathbf{T} = \{U/2\} \\
&\mathcal{C} = \{R_i(X, Y) \to U(X, Y) \mid R_i \in R\} \\
&J = \{U(x, y) \mid (x, y) \in U \times D\} \\
&I = \bigcup_{R_i \in R} \{R_i(x, y) \mid (x, y) \in R_i \times D\}
\end{aligned}
$$

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 $\theta_i = R_i(X, Y) \to U(X, Y)$ 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

$$
\begin{aligned}
F(\mathcal{M}) &= \sum_{t \in J} [1 - \mathsf{explains}_{\mathsf{full}}(\mathcal{M}, t)] + 2 \cdot |\mathcal{M}| \\
&= (m + 1) \cdot \left(|U| - |\bigcup_{\theta_i \in \mathcal{M}} R_i| \right) + 2 \cdot |\mathcal{M}|
\end{aligned}
$$

A mapping $\mathcal{M} \subseteq \mathcal{C}$ with $F(\mathcal{M}) \leq m = 2n$ thus exists if and only if $|\bigcup_{\theta_i \in \mathcal{M}} R_i| = |U|$ and $|\mathcal{M}| \leq n$, 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 (3.4) 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. □

3.4 Mappings with Partial Outputs

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

3.4.1 Incomplete Errors

In contrast to $\mathsf{error}_{\mathsf{full}}(\cdot)$, 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 θ_1 in Figure 3.1d is not valid with respect to the data example in Figure 3.1b. However, if we add the tuple $t_1 = \mathsf{task(BigData, Bob, 123)}$ 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 $k_1 = \mathsf{task(BigData, Bob,}$ $N_0)$ (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 3.1b, 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 3.1d is not valid with respect to the extended data example in Figure 3.1b-(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$(\cdot)$ 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

[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 [54].

helper function:

$$\mathsf{creates}(\theta, t) = \begin{cases} 0 & t \in K_{\mathcal{C}} - J,\, t \notin K_\theta \\ 0 & t \in K_\theta - J,\, \exists h : \mathit{context}_\theta(t) \rightarrow J \\ 1 & t \in K_\theta - J \text{ and no such } h \text{ exists} \end{cases} \tag{3.5}$$

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

$$\mathsf{error}(\mathcal{M}, t) = \sum_{\theta \in \mathcal{M}} \mathsf{creates}(\theta, t) \tag{3.6}$$

In Figure 3.3, which extends Figure 3.2 for selection over st tgds, $\mathsf{error}(\cdot)$ 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 homomorphisms exist from them to 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 t, Equation (3.6) evaluates to 1 if there is no homomorphism from the $\text{context}_\theta(t)$ 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 (3.6) counts how many candidates make this error.

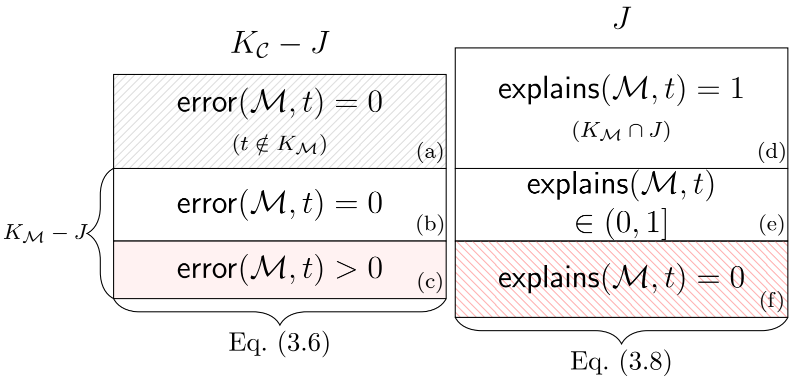

$K_{\mathcal{C}} - J$
J
$\mathsf{error}(\mathcal{M}, t) = 0$
$(t \notin K_{\mathcal{M}})$
(a)
$\mathsf{explains}(\mathcal{M}, t) = 1$
$(K_{\mathcal{M}} \cap J)$
(d)
$K_{\mathcal{M}} - J$
$\mathsf{error}(\mathcal{M}, t) = 0$
(b)
$\mathsf{explains}(\mathcal{M}, t)$
$\in (0, 1]$
(e)
$\mathsf{error}(\mathcal{M}, t) > 0$
(c)
$\mathsf{explains}(\mathcal{M}, t) = 0$
(f)
Eq. (3.6)
Eq. (3.8)

Figure 3.3: Illustration of $\mathsf{explains}(\cdot)$ and $\mathsf{error}(\cdot)$ for selecting st tgds.

3.4.2 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 θ_1 in Figure 3.1d and tuple $t =$task(BigData, Alice, 111) in Figure 3.1b. 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 $h : context_\theta(k) \to J$ 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:

$$\mathsf{covers}(\theta, t) = \begin{cases} \max_{k \in \mathsf{E}(t,\theta)}(1 - \mathsf{null}(k)) & \mathsf{E}(t, \theta) \neq \emptyset \\ 0 & \mathsf{E}(t, \theta) = \emptyset \end{cases} \tag{3.7}$$

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

$$\mathsf{explains}(\mathcal{M}, t) = \max_{\theta \in \mathcal{M}} \mathsf{covers}(\theta, t) \tag{3.8}$$

Equation (3.8) can be used to divide J into three parts (Figure 3.3) 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 $\mathbf{S}$, $\mathbf{T}$, a data example (I, J), and a set $\mathcal{C}$ of candidate st tgds

$$\begin{aligned} \textbf{Find} \quad & \arg\min_{\mathcal{M} \subseteq \mathcal{C}} \sum_{t \in J} [(1 - \mathsf{explains}(\mathcal{M}, t))] \\ & + \sum_{t \in K_{\mathcal{C}} - J} [\mathsf{error}(\mathcal{M}, t)] \\ & + \mathsf{size}(\mathcal{M}) \end{aligned} \tag{3.9}$$

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 3.3, 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.

If all candidates are full, this optimization coincides with the one in (3.4), and so is NP-hard as well (see Section 3.3.4). In Section 3.5, we provide an efficient approximation algorithm for finding a high quality solution $\mathcal{M}$.

3.4.3 Example of Selection over ST TGDs

We extend the running example to illustrate objective (3.9). We use a reduced candidate set $\mathcal{C}' = \{\theta_1, \theta_3\}$ (Figure 3.1d) and the data in Figure 3.1b-(c), but omit the leader relation. A universal solution K_{θ_1} for I contains the task tuples (BigData, Bob, Null$_1$) and (ML, Alice, Null$_2$), while a solution K_{θ_3} contains the task tuples (BigData, Bob, Null$_3$) and (ML, Alice, Null$_4$) and the org tuples (Null$_3$, IBM) and (Null$_4$, SAP).

For θ_1, creates$(\cdot)$ is 1 for tuple task(BigData, Bob, Null$_1$), and 0 for all other tuples, and covers$(\cdot)$ 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 θ_3, creates$(\cdot)$ is 1 for task(BigData, Bob, Null$_3$) and org(Null$_3$,IBM), but 0 for task(ML, Alice, Null$_4$) and org(Null$_4$,SAP), which partially explain task(ML, Alice, 111) and org(111, SAP) to degree $^3/_3$ and $^2/_2$ respectively, via a homomorphism mapping Null$_4$ to 111, with corresponding values for covers$(\cdot)$. The different subsets of candidate st tgds thus obtain the following values for the individual parts and the total of objective function (3.9).

$\mathcal{M}$	$\sum 1 - \text{explains}$	$\sum \text{error}$	size	(3.9)
$\{\}$	4	0	0	4
$\{\theta_1\}$	$3^1/_3$	1	3	$7^1/_3$
$\{\theta_3\}$	2	2	4	8
$\{\theta_1, \theta_3\}$	2	3	7	12

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 θ_3 covers more tuples than θ_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, θ_1 explains each to degree $^2/_3$, and θ_3 fully explains them (while no mapping introduces additional errors).

3.5 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 [44] and statistical relational learning (SRL) [64]. Specifically, CMD encodes the mapping selection objective (Equation (3.9)) as a program in probabilistic soft logic (PSL) [6], 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 $\mathsf{error}(\cdot)$ and $\mathsf{explains}(\cdot)$) by solving a convex optimization problem, while providing theoretical guarantees on solution quality with respect to the combinatorial optimum.

3.5.1 Probabilistic Soft Logic

PSL [6] 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:

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

where $\vec{X}$ is a set of universally-quantified variables, the $b_i(\vec{X})$ and $h_j(\vec{X})$ are atoms over (subsets of) the variables in $\vec{X}$, 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 $\mathcal{I}$ of all ground atoms constructed from the predicates and constants in the program, the truth values of formulas are defined as follows.

$$\mathcal{I}(\ell_1 \wedge \ell_2) = \max\{0, \mathcal{I}(\ell_1) + \mathcal{I}(\ell_2) - 1\}$$

$$\mathcal{I}(\ell_1 \vee \ell_2) = \min\{\mathcal{I}(\ell_1) + \mathcal{I}(\ell_2), 1\}$$

$$\mathcal{I}(\neg l_1) = 1 - \mathcal{I}(\ell_1)$$

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

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

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 $\mathcal{I}$ is:

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

where w_r is the weight of rule r and Z is a normalization constant. PSL inference finds $\arg\max_{\mathcal{I}} f(\mathcal{I})$, that is, the interpretation $\mathcal{I}$ 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 [6].

3.5.2 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 $\mathsf{covers}(\cdot)$ and $\mathsf{creates}(\cdot)$ functions defined in Section 3.4, 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:

$$\mathsf{size}(F): \quad in(F) \rightarrow \bot \tag{3.10}$$

$$1: \quad J(T) \rightarrow \exists F.\ \mathsf{covers}(F,T) \wedge in(F) \tag{3.11}$$

$$1: \quad in(F) \wedge \mathsf{creates}(F,T) \rightarrow J(T) \tag{3.12}$$

Rule (3.10) 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 $\mathsf{size}(f) \cdot (\mathcal{I}(in(f)) - 0)$, and thus minimal if $in(f)$ is false. Rule (3.11) states that if a tuple is in J, there should be an st tgd in the set that covers that tuple, thus implementing the $\mathsf{explains}(\cdot)$ 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 (3.12) 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 $\mathsf{error}(\cdot)$ 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 3.5.3 we show the rules combine to implement the mapping selection objective (3.9).

3.5.3 Objective Equivalence

Recall from Equation (3.9) that our goal is to minimize

$$\sum_{t \in J} [1 - \max_{\theta \in \mathcal{M}} \mathsf{covers}(\theta, t)] \tag{3.13}$$

$$+ \sum_{t \in K_{\mathcal{C}} - J} \left[\sum_{\theta \in \mathcal{M}} \mathsf{creates}(\theta, t) \right] \tag{3.14}$$

$$+ \sum_{\theta \in \mathcal{M}} \mathsf{size}(\theta) \tag{3.15}$$

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 (3.10) for every st tgd $\theta \in \mathcal{C}$:

$$\mathsf{size}(\theta) : in(\theta) \to \bot$$

For $\theta \in \mathcal{M}$, this rule has distance to satisfaction 1, and 0 otherwise. Thus, each $\theta \in \mathcal{M}$

adds $\mathsf{size}(\theta)$ to PSL's distance to satisfaction, so those rules together correspond to (3.15). The error Rule (3.12) 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 $\theta \in \mathcal{C}$:

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

Such a ground rule has distance to satisfaction $\mathsf{creates}(\theta, t) - 0 = \mathsf{creates}(\theta, t)$. Recall from Equation (3.5) 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 (3.14). Rule (3.11) is trivially satisfied for $t \notin J$, and for every $t \in J$ results in a partially ground rule

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

To complete the grounding, we apply PSL's prioritized disjunction rules (Chapter 4). Recall (cf. Section 3.4.2) that the $\mathsf{covers}(\cdot)$ 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

$$1/k : \quad \top \rightarrow \bigvee_{\theta \in \mathcal{C}, \mathsf{covers}(\theta, t) \geq ^i/_k} in(\theta)$$

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 (3.13). Thus, the CMD program optimizes Equation (3.9).

3.5.4 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 $\mathsf{creates}(\cdot)$ atoms (based on Equation (3.5)) and of the $|\mathcal{C}| \times |J|$ many $\mathsf{covers}(\cdot)$ atoms (based on Equation (3.7)). 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.

3.6 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 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.[3]

3.6.1 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 uses the following steps:

1. We generate schemas $\mathbf{S}$ and $\mathbf{T}$, correspondences, the initial data example (I, J_G), and a gold standard mapping $\mathcal{M}_G$ that is valid and fully explaining for (I, J_G) using the metadata generator iBench [43] and data generator ToxGene [65].

2. To create dirty metadata, we generate additional foreign key constraints and correspondences.

3. We use the implementation of the Clio [21] algorithm provided by ++Spicy [23] to generate the set of candidates $\mathcal{C}$ from the schemas, foreign key constraints

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

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 $\mathcal{M}_G$.

The rest of this subsection provides more details on this process, and Table 3.1 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 tgds in $\mathcal{M}_G$. 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 $\pi_{Invocations}$ times. We set the size of I to $\pi_{TuplesPerTable}$ per relation.

Step 2. To obtain candidate st tgds with wrong join paths, we use iBench to add randomly created foreign keys to $\pi_{FKPerc}\%$ of the target relations. To obtain candidate st tgds making wrong connections between the schemas, we introduce additional correspondences as follows. We randomly select $\pi_{Corresp}\%$ 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 [62] can generate $\mathcal{M}_G$

[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.

Parameter	**Range**	**Default**
$\pi_{Invocations}$	1 - 10	2
$\pi_{TuplesPerTable}$	10 - 100	50
π_{FKPerc}	0 - 10%	0%
$\pi_{Corresp}$	0 - 100%	0%
π_{Errors}	0 - 30%	0%
$\pi_{Explainable}$	0 - 100%	0%

Table 3.1: Overview of main experimental parameters.

as part of $\mathcal{C}$). 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 $\mathcal{M}_G$, as unexplainable tuples can be removed prior to optimization (cf. Section 3.3.3). In our scenarios, $\mathcal{M}_G \subseteq \mathcal{C}$, and thus $K_G \subseteq K_{\mathcal{C}}$. So each tuple in $K_{\mathcal{C}}$ is either generated by both $\mathcal{M}_G$ and $\mathcal{C}-\mathcal{M}_G$, only by $\mathcal{M}_G$ (i.e., an error tuple if deleted from J), or only by $\mathcal{C}-\mathcal{M}_G$ (i.e., a new explainable tuple if added to J). As tuples in $K_{\mathcal{C}}$ 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_{New} containing $\pi_{Explainable}$% of the potential new explainable tuples, and a set J_{Err} containing π_{Errors}% of the potential error tuples, and set $J = J_G \cup J_{New} \setminus J_{Err}$.

3.6.2 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. 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 [41]. 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 [61] 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 [42]. We directly use the harmonic mean, which we call $\textsf{IQ-score}(K_1, K_2) \in [0, 1]$, where higher is better.

3.6.3 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 π_{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

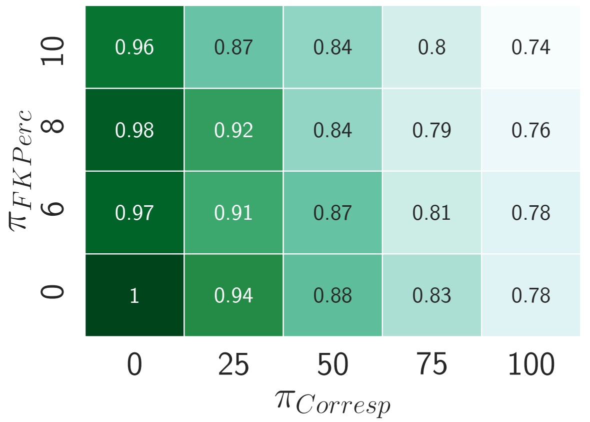


Figure 3.4: Mapping quality (IQ-Score) for the Clio baseline with ambiguous metadata. CMD always reaches IQ-Score 1.

with more imprecise evidence, as shown in Figure 3.4.

3.6.4 CMD Accuracy over Dirty Data

Our second experiment investigates the effect of imperfect data on mapping quality. We vary the percentage π_{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 3.5, we plot IQ-score as we vary π_{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

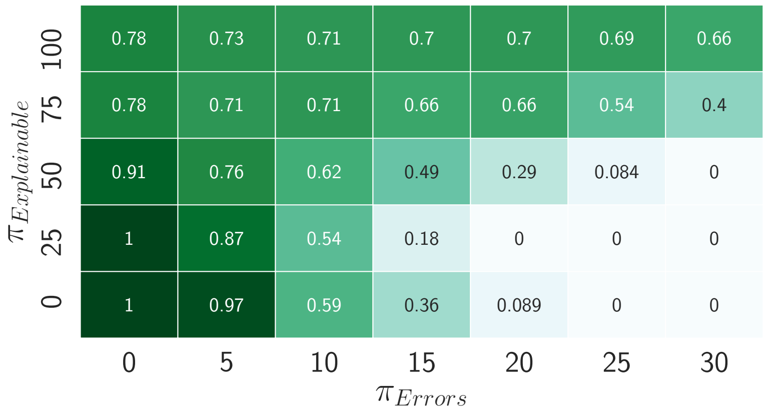
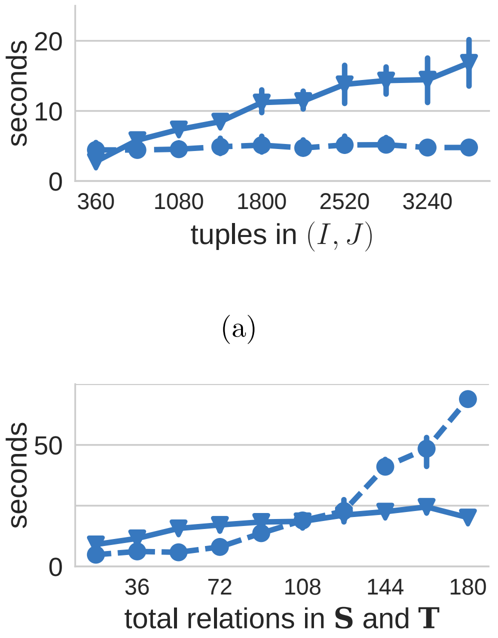


Figure 3.5: Mapping quality (IQ-Score) for CMD with dirty data.

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 $\mathcal{C} - \mathcal{M}_G$ that are similar to $\mathcal{M}_G$, 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.



(a)

(b)

Figure 3.6: Optimization time of CMD w.r.t. (a) size of data example and (b) schema sizes. The dashed line is for the Clio baseline.

3.6.5 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 tgd in $\mathcal{C}$) 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 3.6a plots the average optimization time in

CMD and average time to generate $\mathcal{C}$ (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 3.6b). Again, CMD optimization times are comparable to those of Clio, but for increasing schema size, the latter increases more quickly.

3.6.6 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 [66], 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 [67, 68]. 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.

3.7 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 [5, 21]. HepTox [22] and ++Spicy [23] 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 [25] and in schema mapping, where the Data Viewer [26] and Muse [27] 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 [28]. An example that is closest to ours casts schema mapping discovery from data as a formal (and fully automated) learning problem [32]. 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 [32]. Using this framework, ten Cate et al. [33] 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 [32]), 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 [29].

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. [57] 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 [30]. Finally, Sarma et al. [31] consider how to learn views (or full GAV mappings) from data alone.

Related Selection Problems. Belhajjame et al. [19] 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 they do not provide runtimes for their approach, they use a powerful, general purpose search algorithm [69] 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 [70]. 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. [19] 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 data sources [71], or to select join associations in the context of keyword-search based data integration [72]. 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 [73] 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. [73] 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 [45] for ontology alignment [47] and ontological mapping construction [20]. 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 [58] and data fusion [59, 60].

3.8 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.

Chapter 4: Handling Ambiguity with Prioritized Disjunction Rules

4.1 Introduction

In Chapter 3 we found transformations that explain best the tuples of a relational data example. In Chapter 6, we will encounter a similar problem involving the choice of variable assignments that best explain a correspondence. In both cases, a key challenge is dealing with multiple alternate explanations, e.g., st tgds or correspondences, for data. This type of ambiguity is common in relational integration problems. In fact, even in an ideal example data, there can still be ambiguity due to the nature of the data, e.g., multiple senses for words appearing in tuples, and we are left unsure which transformations are correct. As we have shown, one strategy to solve this important problem is to consider all evidence collectively to see if the aggregate evidence prefers one alternative over others.

A natural way to represent the best alternative in probabilistic logical languages (Section 2.2) is as an existentially-quantified variable plus one or more logical conditions showing that it fits best with the data. For example, Rule (3.11) in Chapter 3 states declaratively that the best alternative single st tgd (variable F) is the one that best covers a target tuple (variable T):

$$1: \quad J(T) \rightarrow \exists F.\ \mathsf{covers}(F, T) \wedge in(F)$$

Similarly, Rule (g) in Chapter 6 states that the best alternative variable assignment is the one that joins attributes matched by a correspondence. However, directly using rules like those can make inference much harder. Specifically, in the language we use – probabilistic soft logic (PSL) [6] – using a conjunction in the head of the rule prevents us from solving inference as a convex optimization problem.

Many existing optimization methods can still be applied to this problem, but they can't provide guarantees of answer quality. Also, some methods do not easily incorporate the soft values of the logical conditions we use in the rules.

I will show that rules of this sort, which we call prioritized disjunction rules (PD), can in fact be rewritten as normal disjunctive clauses, allowing inference in PSL to remain efficient while incorporating the rules. I will show that the new rule type scales well even to highly complex integration problems with many alternative explanations.

4.2 Problem

In first-order logic (with finite domains), formulas with existential quantifiers 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 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 (3.11), over st tgds to be selected). A prioritized disjunction rule is a rule:

$$w: \; b(X) \rightarrow \exists Y.\; h_o(Y,X) \wedge h_i(Y)$$

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.

Example 6: For Rule (3.11) of Chapter 3, $b(X)$ is $J(T)$, $h_o(Y,X)$ is $\mathsf{covers}(F,T)$, and $h_i(Y)$ is $in(F)$.

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 $h_o(Y,X)$. Our goal is to show how to efficiently handle this practically important subclass of rules.

4.3 Approach

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 $\{^0/_k, ..., ^k/_k\}$ 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 $h_o(Y, X) \wedge h_i(Y)$ is either false or equivalent to $h_i(Y)$. Since $h_o(Y, X)$ is observed, for every grounding y of Y, we can drop the entire disjunct if $h_o(y, X)$ is false and 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 $j = 1, .., k$ of the following form:

$$^{w}/_{k} : b(x) \rightarrow \bigvee_{h_o(x,y) \geq {}^{j}/_{k}} h_i(y)$$

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

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

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 $h_o(x, y)$ 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 $j = 1, .., m$ 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 $^{w}/_{k} \cdot \mathcal{I}(b(x))$ to the overall distance to satisfaction, which for this set of ground rules is

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

If b is observed, e.g., $\mathcal{I}(b(x)) = 1$ as in the case of (3.11), this expression depends purely on the maximum value of h_o.

Example 7: Consider a single grounding of Rule (3.11) for $t =$org(111, SAP) in J from Figure 3.1c and the candidates θ_3 and θ_4 from Figure 3.1d. 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 the following possible values for $\mathsf{covers}(\cdot)$:

$$\mathsf{covers}(F, t) \in \{{}^0\!/\!_2, {}^1\!/\!_2, {}^2\!/\!_2\}$$

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 is a useful choice (see Section 3.5.3). While this extension was motivated by relational data integration problems, we expect it to also be useful in other scenarios that involve choices between variable numbers of alternatives.

4.4 Evaluation

To evaluate prioritized disjunction rules, we use the same problem setting described in Chapter 3, and the same basic parameter settings for generated mapping scenarios. The evaluation in Section 3.6 tested Rule (3.11) on a variety of mapping scenarios generated using iBench [43]. In this chapter, we test the effectiveness and scalability of prioritized disjunction rules on an additional, novel type of mapping scenario with much more complex sets of alternative explanations, and having a new parameter for varying that complexity.

4.4.1 Complex Scenario Generation

In this section, we describe the custom-made scenarios we use to test prioritized disjunction rules. In each scenario, the source schema consists of $\pi_{SchemaSize}$ relations S_1 to S_m, all of arity π_{Arity}. The target schema consists of $\pi_{SchemaSize}$ relations T_1 to T_m, all of arity π_{Arity}, and a set of relations R_i^j of arity j, where $1 \leq i \leq \pi_{SchemaSize}$ and $1 \leq j \leq \pi_{Arity} - 1$. Using these, we construct a space $\mathcal{A}$ of candidate st tgds, where $1 \leq a \leq b \leq \pi_{Arity}$, as follows:

$$\begin{aligned} S_i(X_1, \ldots, X_a, Y_{a+1}, \ldots, Y_k) \rightarrow & \exists X_{a+1} \ldots X_k. \\ & T_j(X_1 \ldots X_k) \wedge \\ & R_j^{k-b}(X_{b+1} \ldots X_k) \end{aligned}$$

That is, we pair each source relation S_i with each target relation T_j. The pair (a, b) determines which arguments of T_j are exchanged (X_1 to X_a, always at least one), unshared existentials (X_{a+1} to X_b, potentially empty) and used for the target join with the appropriate R_j^l (X_{b+1} to X_k, potentially empty, in which case R_j^l is omitted). For instance, if $k = 3$, for one pair of S and T, we get the following st tgds, which explain T to degree $^3/_3$, $^2/_3$, $^1/_3$, $^3/_3$, $^2/_3$ and $^3/_3$, respectively.

$$S(X,A,B) \to T(X,Y,Z) \wedge R(Y,Z)$$

$$S(X,A,B) \to T(X,Y,Z) \wedge R(Z)$$

$$S(X,A,B) \to T(X,Y,Z)$$

$$S(X,Y,B) \to T(X,Y,Z) \wedge R(Z)$$

$$S(X,Y,B) \to T(X,Y,Z)$$

$$S(X,Y,Z) \to T(X,Y,Z)$$

To construct the candidates $\mathcal{C}$ for a particular scenario, we randomly pick from $\mathcal{A}$, for each $i = 1, \ldots, \pi_{SchemaSize}$, one of the candidates that combine S_i and T_i (and thus also some R_i^j; we refer to this st tgd as θ_i) and add it to the gold standard mapping $\mathcal{M}_G$. Thus, $|\mathcal{M}_G| = \pi_{SchemaSize}$. From $\mathcal{A} \backslash \mathcal{M}_G$, for each $i = 1, \ldots, \pi_{SchemaSize}$, we randomly pick $\pi_{PDSize} - 1$ of the candidates that use T_i. We obtain $\mathcal{C}$ by adding these to $\mathcal{M}_G$; thus, $|\mathcal{C}| = \pi_{PDSize} \cdot \pi_{SchemaSize}$. As $\mathcal{A}$ contains $m\frac{k(k+1)}{2}$ st tgds using T_i, we can scale π_{PDSize} to at most that number (in which case $\mathcal{C} = \mathcal{A}$). $\mathcal{A}$ by construction contains candidates covering the T-relations to each non-zero degree.

To construct the data example for a particular scenario, we construct separate *private* and *common* sets of tuples. First, for each θ_i (using S_i and T_i) in the gold standard mapping, we create $\pi_{Private}$ private π_{Arity}-tuples, which we add to T_i in J. Furthermore, if θ_i contains a join, we also add the corresponding sub-tuples to the corresponding R_i-relation in J. Finally, to populate S_i, in each case, we create a

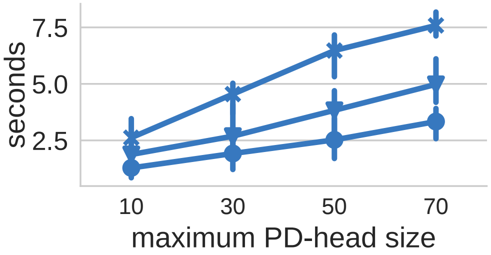


Figure 4.1: Optimization time of CMD w.r.t. maximal number of candidates explaining the same tuple. Parameter $\pi_{SchemaSize}$ is 20 (top), 10 (middle) or 5 (bottom).

source tuple that shares the exchanged arguments with the one in T_i. For instance, for $S(X, A, B) \rightarrow T(X, Y, Z) \wedge R(Z)$, we could get tuples $T(a1, b1, c1)$, $R(c1)$ and $S(a1, b2, c2)$. Second, we generate π_{Common} common π_{Arity}-tuples, and we add those to every relation S_i and every relation T_i, as well as the corresponding suffixes to every relation R_i^l. For instance, one such step could use $(a13, b13, c13)$ to add all $T_i(a13, b13, c13)$, $S_i(a13, b13, c13)$, $R_i^2(b13, c13)$ and $R_i^1(c13)$. This common part of the data example introduces confusion into the selection process.

4.4.2 Results

We vary the maximal number π_{PDSize} of candidate st tgds that explain each target tuple, which corresponds to the number of head atoms in prioritized disjunction rules (see Chapter 4). 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 π_{Arity} and the schema size $\pi_{SchemaSize}$. The source has $\pi_{SchemaSize}$ relations of arity π_{Arity}, the target $\pi_{SchemaSize} \cdot \pi_{Arity}$ relations of arity at most π_{Arity}. The gold standard mapping has $\pi_{SchemaSize}$ st tgds, and the number of potential candidates increases quadratically with both $\pi_{SchemaSize}$ and π_{Arity}. We set $\pi_{TuplesPerTable} = 25$.

We consider all combinations of $\pi_{SchemaSize} \in \{5, 10, 20\}$ and $\pi_{Arity} \in \{5, 10, 20\}$, and vary π_{PDSize} from 10 to 70 in steps of 20. We use one scenario for each combination. In Figure 4.1, we plot the optimization time for each value of $\pi_{SchemaSize}$, aggregating over π_{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.

4.5 Conclusion

In this chapter, we introduced a novel extension to PSL called prioritized disjunction rules, which allows for existentials over closed domains while maintaining the convexity of inference. We demonstrate the effectiveness of this type of rule in the mapping selection problem setting. We show that on problems with highly complex sets of alternate explanations, it efficiently selects the correct alternatives.

Chapter 5: Mapping Search

5.1 Introduction

In previous chapters we described how we can derive possible transformations from metadata, and combine them with data in a probabilistic, collective framework to handle noise, ambiguity and partial outputs. However, ideal transformations may not be derivable from metadata because parts of the metadata are very noisy or even missing. Transformations found this way can have some correct portions, but also many *flaws*. In practice, many sources are available only in this highly raw form, so it is critical to develop methods that discover good transformations even in this setting.

As we did in Chapter 3, we can leverage a holistic combination of metadata and data whenever both are available. Then we can set up an exhaustive search over possible transformations, repeatedly changing parts and checking whether the changes improve the fit to the evidence. However, the space of possible st tgd mappings is very large for three reasons: First, the body of a st tgd can have atoms for any or all of the relations of the source schema. The same is true for the head of a st tgd and the target schema relations, respectively. Second, logical variables can repeat across arguments of an atom – and across multiple atoms of the body or head or both – in any arrangement. Finally, recall that a mapping is a set of st tgds and – as we found in Chapter 3 – selecting a subset from a set of given st tgds is already an intractable problem if handled naively. Therefore, an unguided search over possible transformations is only practical with exceptionally small schemas.

There has been significant past work – in both theory and practical systems – on closely related problems, such as theory refinement [49]. However, most existing work focuses on learning single target relations, e.g., clauses, and does not consider the important effect of existential variables shared across multiple target relations.

In this chapter, we will introduce a new approach to guide a search over possible mappings, exploiting a useful property of the objective we introduced in Chapter 3 that allows us to use it to guide both selection over subsets of st tgds and refinement of individual st tgds. Specifically, we use a *boosted search* organized in stages. In each stage, we change the makeup of the mapping – suppressing st tgds, generating new individual st tgds, or both. We generate new st tgds using refinement operators, guided via errors caused by flaws of the current mapping.

We evaluate this approach on a diverse collection of parameterized mapping scenarios with varying complexity in each of the three dimensions that make search spaces large: (1) varying atoms in the body and head of st tgd, (2) varying the assignments of logical variables in arguments, and (3) varying the subset of st tgds that should be selected. We show that, compared to a baseline using conventional single-relation learning, our approach achieves over 30% higher mapping quality over-all. We also compare to the selection-only capability introduced in Chapter 3, showing that it improves mapping quality by over 80%. Additionally, we show that our approach finds quality mappings with greater consistency than both baselines. Finally, we test our approach on a real data problem and show that it discovers correct st tgds even with very limited metadata.

This chapter is organized as follows: In Section 5.2, we describe the mapping search problem. In Sections 5.3, 5.4, and 5.5 we introduce our search approach, the refinement operators we use, and the baseline algorithms, respectively. In Sections 5.6 and 5.7, we describe our scenario generation process and our empirical evaluation results. In Sections 5.8 and 5.9, we describe related work and some possible future extensions of this work.

5.2 Mapping Search Problem

In this section, we describe the mapping search problem. First, we review measures of mapping quality from earlier work. Then we introduce an improvement to the mapping quality definition motivated by the mapping search setting. Finally, we define the mapping search problem formally.

5.2.1 Mapping Quality

In this section, we review measures of mapping quality introduced in Chapter 3. We use the data exchange setting to characterize the quality of mappings comprising one or more st tgds. In this setting, we are given a data example comprising a source instance I and a target instance J. Given a source instance and some mapping, we can generate a target instance known as an *exchange solution.* To measure mapping

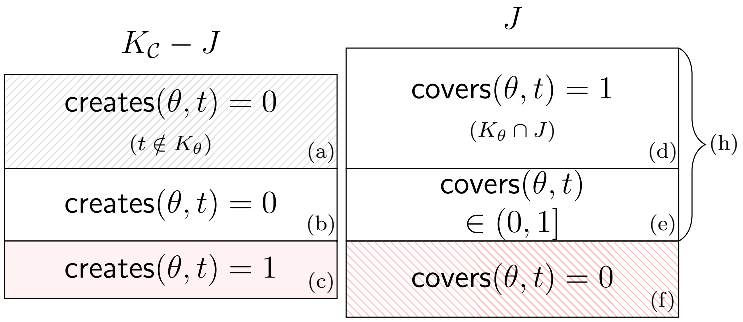

$K_\mathcal{C} - J$
J
$\mathsf{creates}(\theta, t) = 0$
$(t \notin K_\theta)$
(a)
$\mathsf{creates}(\theta, t) = 0$
(b)
$\mathsf{creates}(\theta, t) = 1$
(c)
$\mathsf{covers}(\theta, t) = 1$
$(K_\theta \cap J)$
(d)
$\mathsf{covers}(\theta, t)$
$\in (0, 1]$
(e)
$\mathsf{covers}(\theta, t) = 0$
(f)
(h)

Figure 5.1: Illustration of $\mathsf{covers}(\cdot)$ and $\mathsf{creates}(\cdot)$.

quality, we compare the following:

- Tuples of J
- Tuples of the solution for I and a mapping

We use the comparison to measure two kinds of flaws: *Errors* are tuples in a solution with no corresponding tuple in J, i.e., false positives. *Unexplained* tuples are the second kind of error, and arise when a tuple of J has no corresponding tuple in the solution, i.e., false negatives.

Our formal definitions for the two kinds of errors take one form when evaluating individual st tgds, and another form when we evaluate a set of st tgds collectively. Although a single st tgd alone may be used as a mapping, for clarity we only use the term *mapping* when referring to a set of st tgds. Figure 5.1 illustrates the case for individual st tgds. If a tuple t is an error with respect to st tgd θ, we represent that fact as $\mathsf{creates}(\theta, t) = 1$ (Figure 5.1 (c)). If t is unexplained with respect to θ, we represent that as $\mathsf{covers}(\theta, t) < 0$ (Figure 5.1 (e) and (f)).

Figure 5.2 illustrates the case for a mapping $\mathcal{M}$. If t is an error with respect to $\mathcal{M}$, we say $\mathsf{error}(\mathcal{M}, t) > 0$ (Figure 5.2 (c)). If t is unexplained with respect to $\mathcal{M}$,

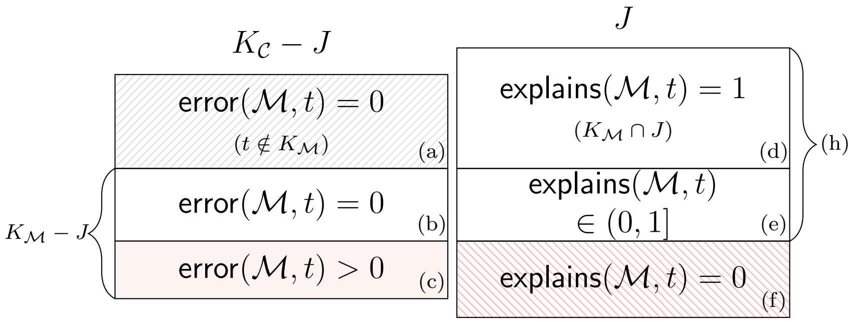

$K_{\mathcal{C}} - J$
J
$\mathsf{error}(\mathcal{M}, t) = 0$
$(t \notin K_{\mathcal{M}})$
(a)
$\mathsf{explains}(\mathcal{M}, t) = 1$
$(K_{\mathcal{M}} \cap J)$
(d)
(h)
$K_{\mathcal{M}} - J$
$\mathsf{error}(\mathcal{M}, t) = 0$
(b)
$\mathsf{explains}(\mathcal{M}, t) \in (0, 1]$
(e)
$\mathsf{error}(\mathcal{M},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

Figure 5.2: Illustration of $\mathsf{explains}(\cdot)$ and $\mathsf{error}(\cdot)$ for selecting st tgds.

we say $\mathsf{explains}(\mathcal{M}, t) < 0$ (Figure 5.2 (e) and (f)).

In addition to errors and explains, we are also interested in a mapping that is simple. A mapping that achieves high quality but is highly complex may be over-fit on the data example. We measure the complexity of a mapping using function size.

5.2.1.1 Revised Covers

We revise the definition of covers for st tgds generating nulls, compared to Chapter 3. Recall that, because st tgd can create nulls, we use logical homomorphisms as an initial check for whether a st tgd can cover a target tuple. If a homomorphism exists, we calculate the degree of coverage by inspecting the arguments of the generated tuple. As before, we recognize two ways that a st tgd covers a argument: (a) The st tgd provides a constant from I to fill the argument in the target solution. (b) The st tgd provides a null with a label, and it places the same label on nulls in other arguments. Shared labels on nulls are useful because they establish a relational structure and put constraints on values that could be invented for the argument in the solution. As before, when a label appears in only one argument, we treat that argument as a regular null, contributing a value of one to the aggregate count of nulls. At the other extreme is a argument receiving a constant, which contributes a zero to that sum.

Our revised definition of covers is motivated by the simple observation that if correctly reusing a null once is useful, then additional correct reuse is even more useful. We change the value contributed by arguments with a labeled null if the label appears in two or more arguments. Every argument now contributes a value between zero and one to the count of nulls, where values close to zero are possible if a label appears in may arguments. Specifically, the value is simply the reciprocal of the number of arguments in which a null appears. Our previous definition for the null function was piecewise and Boolean: labels appearing in just two or more arguments had value zero, i.e., the same as a constant. The revised definition recognizes that as a st tgd places the same label in more arguments, it places more constraints on the nulls in those arguments, i.e., approaching the type of full constraint provided by a constant. The reciprocal is one of many possible definitions with this characteristic; we chose it for its simplicity.

5.2.2 Search Objective

In this section we give a formal definition for the mapping search problem. Each mapping $\mathcal{M}$ is a set of st tgds of the following form: $\forall \vec{x}, \vec{z}.\ \phi(\vec{x}, \vec{z}) \rightarrow \exists \vec{y}\ \psi(\vec{y}, \vec{z})$, where $\phi(\vec{x}, \vec{z})$ and $\psi(\vec{y}, \vec{z})$ are conjunctions of atoms. We define $\mathsf{refinements}(\mathcal{C}, \boldsymbol{\rho})$ as the set of candidates such that $\theta' \in \mathsf{refinements}(\mathcal{C}, \boldsymbol{\rho})$ if θ' can be generated from some $\theta \in \mathcal{C}$ by applying a sequence of refinement operators from set $\boldsymbol{\rho}$ (see Section 5.4).

Our mapping search objective is the following:

Given

- schemas $\mathbf{S}$, $\mathbf{T}$, and a data example (I, J)
- initial set $\mathcal{C}$ of candidate st tgds, which may be empty
- a set of operators $\boldsymbol{\rho}$

$$\textbf{Find} \quad \begin{aligned} \mathop{\arg\min}_{\mathcal{M} \subseteq \mathsf{refinements}(\mathcal{C}, \boldsymbol{\rho})} & \sum_{t \in J} [(1 - \mathsf{explains}(\mathcal{M}, t))] \\ & + \sum_{t \in K_{\mathcal{C}} - J} [\mathsf{error}(\mathcal{M}, t)] \\ & + \mathsf{size}(\mathcal{M}) \end{aligned} \tag{5.1}$$

Intuitively, our objective is to find a set of st tgds that maximizes explained tuples, that minimizes error tuples, and that is not very complex. This is a natural extension of Equation (3.9), with the addition of candidates that can be generated with refinement operators.

Monotonicity. An important property of the objective is that the value we minimize decreases monotonically with repeated expansions of $\mathsf{refinements}(\mathcal{C}, \boldsymbol{\rho})$ by applying operators in $\boldsymbol{\rho}$. This is true simply because adding a st tgd to the set of candidate refinements does not place that st tgd in the mapping $\mathcal{M}$. Only st tgds in $\mathcal{M}$ affect the terms of the objective, and – with a guarantee of quality – a subsequent selection step will only add a refinement to $\mathcal{M}$ if it further optimizes the mapping. The objective assumes we have access to all refinements, which is a set that can be much larger than practical. The monotonicity property allows us to approach the optimal by alternating between expanding the set of available refinements and selecting the best subset. This will be the basis for our search approach in Section 5.3.

5.3 Search Approach

In this section, we propose an approach to approximate the objective in (5.1) with a search algorithm. As motivated by the monotonicity property of that objective, our algorithm alternates between expanding the set of available st tgds, and selecting an optimal subset as the mapping. At a high level, each stage of the algorithm involves the following three steps:

1. Identify flaws of the current mapping

2. Generate additional candidate st tgds

3. Select a new subset of candidate st tgds

Step 1 uses the quality measures from Section 5.2.1 to identify target relations and individual st tgds that are flawed. Step 2 then builds a set of new candidate st tgds, guided by the flaws found in step 1. Focusing learning only on errors made by the current hypothesis is sometimes called *boosting* and is a common technique. Step 3 then selects a new optimal mapping as a subset of the set of candidates; this step is identical to mapping selection in Chapter 3. In summary, in each stage we can change the makeup of the mapping: suppressing st tgds, generating new individual st tgds, or both.

Symbol	Meaning
I, J	Given source and target instances
$\mathbf{S}, \mathbf{T}$	Source and target schemas
T	A relation from $\mathbf{T}$
$J(T)$	Subset (tuples) of J in T
θ	A st tgd
$\theta^{(-1)}$	Original version of θ, before refinement
$\Theta^{(-1)}$	Set of original st tgds
$\mathcal{C}^{(0)}$	Given set of candidates
$\mathcal{C}^{(m)}$	Candidates from current stage
$\mathcal{C}^{(-1)}$	Candidates from previous stage $m-1$
$\mathcal{M}$	Set of st tgds forming a mapping
$\mathcal{M}^{(0)}$	A mapping selected from $\mathcal{C}^{(0)}$
$\mathcal{M}^{(m)}$	Mapping from current stage
$\mathcal{M}^{(-1)}$	Mapping from previous stage $m-1$
$\mathcal{M}^{(M)}$	Mapping from final stage
$\mathcal{M}(T)$	Subset (st tgds) of $\mathcal{M}$ that fill relation T
$\mathcal{M}^{(-1)}(T)$	Subset of $\mathcal{M}^{(-1)}$ that fill relation T
K_θ	Solution using I and θ
$K_\theta^{(-1)}$	Solution using I and $\theta^{(-1)}$
$K_\theta^{(-1)}(T)$	Subset (tuples) of $K_\theta^{(-1)}$ in T
$K_\mathcal{M}^{(m)}$	Solution using I and $\mathcal{M}^{(m)}$
$K_\mathcal{M}^{(-1)}$	Solution using I and $\mathcal{M}^{(-1)}$
$K_\mathcal{M}^{(-1)}(T)$	Subset of $K_\mathcal{M}^{(-1)}$ in T
$K_\mathcal{C}$	Solution using I and $\mathcal{C}^{(m)}$
$K_\mathcal{C}^{(-1)}$	Solution using I and $\mathcal{C}^{(-1)}$
$K_\mathcal{C}^{(-1)}(T)$	Subset of $K_\mathcal{C}^{(-1)}$ in T
$\mathsf{creates}_\theta$	$\mathsf{creates}(\theta, t)$
$\mathsf{creates}_\theta^{(-1)}$	$\mathsf{creates}(\theta^{(-1)}, t)$
covers_θ	$\mathsf{covers}(\theta, t)$
$\mathsf{covers}_\theta^{(-1)}$	$\mathsf{covers}(\theta^{(-1)}, t)$
$\mathsf{error}_\mathcal{M}$	$\mathsf{error}(\theta, t)$
$\mathsf{error}_\mathcal{M}^{(-1)}$	$\mathsf{error}(\theta^{(-1)}, t)$
$\mathsf{explains}_\mathcal{M}^{(m)}$	$\mathsf{explains}(\mathcal{M}^{(m)}, t)$
$\mathsf{explains}_\mathcal{M}^{(-1)}$	$\mathsf{explains}(\mathcal{M}^{(-1)}, t)$
$\mathsf{explains}_{\mathcal{M}-\theta}^{(-1)}$	$\mathsf{explains}(\mathcal{M}^{(-1)} - \theta^{(-1)}, t)$
$s^{(-1)}$	$s(\theta^{(-1)}, \theta)$, the refinement score

Table 5.1: Notation and abbreviations.

Algorithm 1: Search function $\mathcal{S}_{\texttt{sib}}$.

```
Data:
Initial candidate set: $\mathcal{C}^{(0)}$
Refinement operator set: $\boldsymbol{\rho}$
Search stages: $M$
Refinement iterations: $U$
Refinement score: $s$
Result: $\mathcal{M}$
$\mathcal{M}^{(0)} \leftarrow$ MappingSelection($\mathcal{C}^{(0)}$);
for $m$ from 1 to $M$ do
    $\mathcal{C}^{(m)} \leftarrow \mathcal{C}^{(\text{-}1)}$;
    Select flawed relation $T \in \mathbf{T}$;
    Stop if total for $T$ in Equation (5.2) is zero;
    Based on $T$, select st tgd set $\Theta^{(\text{-}1)}$;
    for $\theta^{(\text{-}1)} \in \Theta^{(\text{-}1)}$ do
        Update weights used by score function $s^{(\text{-}1)} = s(\theta^{(\text{-}1)}, \theta)$;
        $\theta \leftarrow$ Refine($\theta^{(\text{-}1)}, \mathcal{C}^{(\text{-}1)}, s^{(\text{-}1)}, \boldsymbol{\rho}, U$);
        $\mathcal{C}^{(m)} \leftarrow \mathcal{C}^{(m)} \cup \{\theta\}$;
    end
    $\mathcal{M}^{(m)} \leftarrow$ MappingSelection($\mathcal{C}^{(m)}$);
end
return $\mathcal{M}^{(M)}$;
```

We now define the algorithm formally. As shown in Algorithm 1, search function $\mathcal{S}_{\texttt{sib}}$ starts with an initial set of candidates. We refer to the initial set as $\mathcal{C}^{(0)}$ because it is the set prior to the first stage of the search. The algorithm then expands the candidates over M stages using set $\boldsymbol{\rho}$ of refinement operators. For each stage m, we refer to candidates of previous stage $m-1$ as $\mathcal{C}^{(m-1)}$. For brevity, where the context is clear we will drop m and shorten $\mathcal{C}^{(m-1)}$ to $\mathcal{C}^{(\text{-}1)}$. Similarly, we refer to previous versions of mappings and st tgds as $\mathcal{M}^{(\text{-}1)}$ and $\theta^{(\text{-}1)}$, respectively. We summarize all notation in Table 5.1.

5.3.1 Step 1.1: Select a Flawed Relation

In each stage, we focus the search on a single target relation for which the current mapping is a poor fit. Before doing that, we first select the best available mapping from $\mathcal{C}^{(\text{-}1)}$. Function MappingSelection($\mathcal{C}$) selects an optimal subset $\mathcal{M}^{(\text{-}1)}$ of set $\mathcal{C}^{(\text{-}1)}$ of candidate st tgds (line 1); for this we use CMD (see Chapter 3). To select (line 4) a target relation $T \in \mathbf{T}$, we prefer a relation having many tuples left to explain or errors to correct. We refer to tuples of relation T in instance J as $J(T)$ and those in solution $K_{\mathcal{M}}^{(\text{-}1)}$ of $\mathcal{M}^{(\text{-}1)}$ as $K_{\mathcal{M}}^{(\text{-}1)}(T)$. We find an optimal relation T as follows:

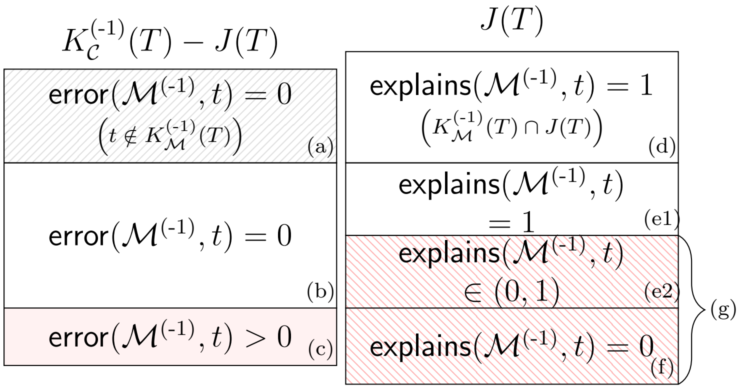

$K_{\mathcal{C}}^{(\text{-}1)}(T) - J(T)$
$J(T)$
$\mathsf{error}(\mathcal{M}^{(\text{-}1)}, t) = 0$
$\left(t \notin K_{\mathcal{M}}^{(\text{-}1)}(T)\right)$
(a)
$\mathsf{error}(\mathcal{M}^{(\text{-}1)}, t) = 0$
(b)
$\mathsf{error}(\mathcal{M}^{(\text{-}1)}, t) > 0$
(c)
$\mathsf{explains}(\mathcal{M}^{(\text{-}1)}, t) = 1$
$\left(K_{\mathcal{M}}^{(\text{-}1)}(T) \cap J(T)\right)$
(d)
$\mathsf{explains}(\mathcal{M}^{(\text{-}1)}, t)$
$= 1$
(e1)
$\mathsf{explains}(\mathcal{M}^{(\text{-}1)}, t)$
$\in (0, 1)$
(e2)
$\mathsf{explains}(\mathcal{M}^{(\text{-}1)}, t) = 0$
(f)
(g)

Figure 5.3: Illustration of $\mathsf{explains}(\cdot)$ and $\mathsf{error}(\cdot)$ for $T \in \mathbf{T}$ and $\mathcal{M}^{(\text{-}1)} \in \mathcal{C}^{(\text{-}1)}$, as used in Equation (5.2).

$$\begin{aligned} \underset{T \in \mathbf{T}}{\arg\max} \quad & \sum_{t \,\in\, J(T)} \left[1 - \mathsf{explains}\left(\mathcal{M}^{(\text{-}1)}, t\right)\right] \\ + & \sum_{t \in K_{\mathcal{M}}^{(\text{-}1)}(T) - J(T)} \left[\mathsf{error}\left(\mathcal{M}^{(\text{-}1)}, t\right)\right] \end{aligned} \tag{5.2}$$

The $\mathsf{explains}(\cdot)$ term of Equation (5.2) calculates the total amount that tuples of T in the data example remain unexplained by the previous mapping $\mathcal{M}^{(\text{-}1)}$. That quantity includes the number of completely unexplained tuples (illustrated in Figure 5.3(f)) and the amount that partially explained tuples remain unexplained (Figure 5.3(e2)). The $\mathsf{error}(\cdot)$ term is the number of error tuples of relation T made by $\mathcal{M}^{(\text{-}1)}$, shown in Figure 5.3(c). That is, we select T to maximize (c) and (g) of Figure 5.3.

A common alternative technique is to simply select each target relation in turn and learn a rule for each. However, in some mapping problems this will not work well. For example, suppose a target relation is formed from the union of tuples from multiple source relations. Selecting that target relation once, or a fixed number of times, may not generate st tgds for each of the source relations. Our approach allows the search to repeatedly select the same relation until all its tuples are explained.

5.3.2 Step 1.2: Select Flawed ST TGDs

Based on T, we then construct set $\Theta^{(\text{-}1)}$ of st tgds to refine (line 6), as follows. We construct the set to include three types of st tgd:

- Existing st tgds that generate tuples of T, which may be flawed
- Existing st tgds modified to have T in their head
- A new st tgd with just T in its head

We define the set $\mathcal{M}^{(\text{-}1)}(T) \subseteq \mathcal{M}^{(\text{-}1)}$ as the set of st tgds generating tuples of T, i.e., with the relation T appearing in the head. We create a second set $\mathcal{M}^{(\text{-}1)}(T)'$ from the remaining st tgds $\mathcal{M}^{(\text{-1})} - \mathcal{M}^{(\text{-1})}(T)$. We modify this set as follows to have T in their heads, where the notation $\lambda\{u/v\}$ denotes the formula derived from λ by substituting all occurrences of u with v.

$$\rho(\theta) = \Big\{ \big\{ \phi(\vec{x}, \vec{z}) \rightarrow \exists \vec{y}.\ \psi(\vec{y}, \vec{z}) \wedge T(\vec{w}) \{w_j / y_k\} \text{ with } w_j \in \vec{w}, y_k \in \vec{y} \big\} \cup$$
$$\big\{ \phi(\vec{x}, \vec{z}) \rightarrow \exists \vec{y}.\ \psi(\vec{y}, \vec{z}) \wedge T(\vec{w}) \{w_j / z_k\} \text{ with } w_j \in \vec{w}, z_k \in \vec{z} \big\} \Big\}$$

This kind of function is widely used in search algorithms, and is known as a *refinement* operator. We define $\mathcal{M}^{(\text{-1})}(T)'$ as follows:

$$\mathcal{M}^{(\text{-1})}(T)' = \bigcup_{\theta \in \mathcal{M}^{(\text{-1})} - \mathcal{M}^{(\text{-1})}(T)} \rho(\theta)$$

An advantage of using set $\mathcal{M}^{(\text{-1})}(T)'$ is it allows us to gradually build highly expressive st tgds combining multiple target relations. The full set of st tgds to refine is as follows:

$$\Theta^{(\text{-1})} = \mathcal{M}^{(\text{-1})}(T) \cup \mathcal{M}^{(\text{-1})}(T)' \cup \{\rightarrow T(\ldots)\}$$

That is, we refine every st tgd currently generating tuples of T (those st tgds may be refined to explain more T tuples, to explain T more fully, or corrected to avoid

T errors); we modify all other st tgds in the current mapping to generate T, then refine them; and we start and refine new st tgds for T from scratch. This is not the only approach to build set $\Theta^{(\text{-}1)}$; other approaches may be more efficient. We chose this simple approach – which includes all existing st tgds in $\mathcal{M}$ and one from scratch – to provide many possible refinements to later steps. An aggressive approach like this is possible because the selection step (line 12) scales well to large numbers of candidates (see Sections 3.6.5 and 4.4.2).

5.3.3 Step 2.1: Refinement Objective

Generating all refinements of the st tgds in $\Theta^{(\text{-}1)}$ would be impractical; instead, we search for refinements that fix flaws in those st tgds. To do this, we define a refinement objective is as follows:

Given

- schemas $\mathbf{S}$, $\mathbf{T}$, and a data example (I, J)
- current mapping $\mathcal{M}^{(\text{-}1)}$
- initial st tgd $\theta^{(\text{-}1)}$
- a set of operators $\boldsymbol{\rho}$

Find $$\underset{\theta \in \mathsf{refinements}(\theta^{(\text{-}1)}, \boldsymbol{\rho})}{\arg\max} \quad s(\theta^{(\text{-}1)}, \theta) \tag{5.3}$$

The objective is based on the following refinement score:

$$s(\theta^{(\text{-}1)}, \theta) = \sum_{\substack{t \in J : \left(\mathsf{explains}^{(\text{-}1)}_{\mathcal{M}} < 1\right) \\ \vee \left(\mathsf{explains}^{(\text{-}1)}_{\mathcal{M}-\theta} < \mathsf{covers}^{(\text{-}1)}_{\theta}\right)}} [\mathsf{covers}(\theta, t)] \\ - \sum_{t \in K_\theta - J} [\mathsf{creates}(\theta, t)] \tag{5.4}$$

Intuitively, the refinement score measures the potential improvement to the overall objective from using a refinement θ instead of the original $\theta^{(\text{-}1)}$. In the remainder of this section, we explain Equation (5.4) and show how we derive it from the overall objective Equation (5.1).

The data example (I, J) is fixed and used by all stages. However, we use the quality of the mapping $\mathcal{M}^{(\text{-}1)}$ (see Section 5.2.1) from the previous stage to weight the data example during the current stage, a common boosting technique. We achieve a weighting over example target tuples J by omitting some tuples from the refinement score function $s(\theta^{(\text{-}1)}, \theta)$ (line 8); for brevity, we use $s^{(\text{-}1)}$. That is, all weights are either zero or one. Using weights other than 0 or 1 is a potential future research problem.

We define $s^{(\text{-}1)}$ over refinement θ of $\theta^{(\text{-}1)} \in \mathcal{M}^{(\text{-}1)}$ based on the potential of θ to change the sum in Equation (5.1). To motivate this choice, consider a comparison with approaches using theory operators [74, 75]. A theory operator γ_ρ uses an underlying rule operator ρ as follows, where $\mathcal{L}_{tgd}$ is the language of st tgds:

$$\begin{aligned}\gamma_\rho(\mathcal{M}^{(\text{-}1)}) = &\Big\{\mathcal{M}^{(\text{-}1)} - \{\theta^{(\text{-}1)}\} \cup \{\theta\} \text{ with} \\ &\quad \theta \in \rho(\theta^{(\text{-}1)}) \text{ and } \theta^{(\text{-}1)} \in \mathcal{M}^{(\text{-}1)}\Big\} \cup \\ &\Big\{\mathcal{M}^{(\text{-}1)} \cup \{\theta\} \text{ with } \theta \in \mathcal{L}_{tgd}\Big\}\end{aligned}$$

A theory operator generates refinements of whole hypotheses by either replacing an existing rule or adding a new rule. Similarly, in our approach MappingSelection may replace an existing rule with a refinement or add a new rule from scratch. MappingSelection has additional flexibility to select both or neither. However, score $s^{(\text{-}1)}$ over refinements of the underlying operator ρ need only consider two situations: Compare a refinement θ to the original $\theta^{(\text{-}1)} \in \mathcal{M}^{(\text{-}1)}$. Or, if the refinement was created from scratch, compare to the empty mapping, i.e., $\theta^{(\text{-}1)} = \{\}$, having $\mathsf{creates}(\cdot) = \mathsf{covers}(\cdot) = 0$ for all target tuples. In either situation, we focus on the following two changes:

- Increased tuple *explanation* (Figure 5.2 (h))
- Decreased tuple *errors* (Figure 5.2 (c))

In the next sections, we define both kinds of changes and how we combine them in the score.

covers_θ ? $\mathsf{covers}_\theta^{\text{(-1)}}$	Eq. 5.5 < 0	Eq. 5.5 $= 0$	Eq. 5.5 > 0
$>$	never	(a) $\mathsf{covers}_\theta \leq$ $\mathsf{explains}_{\mathcal{M}}^{\text{(-1)}} \leq 1$	(b) $\mathsf{explains}_{\mathcal{M}}^{\text{(-1)}} <$ $\mathsf{covers}_\theta \leq 1$
$=$	never	always	never
$<$	(c) $\mathsf{explains}_{\mathcal{M}-\theta}^{\text{(-1)}} <$ $\mathsf{covers}_\theta^{\text{(-1)}}$	(d) $\mathsf{covers}_\theta^{\text{(-1)}} \leq$ $\mathsf{explains}_{\mathcal{M}-\theta}^{\text{(-1)}}$	never

Table 5.2: Effects of refinement on $\mathsf{explains}(\cdot)$ (see Equation (5.5)).

5.3.3.1 Explains Boosting

Replacing a st tgd in $\mathcal{M}$ with a refinement makes the following change to the $\mathsf{explains}(\cdot)$ term of the overall objective for some tuple, where for brevity we replace $\mathsf{explains}(\mathcal{M}^{(m)}, t)$ with $\mathsf{explains}_{\mathcal{M}}^{(m)}$; replace $\mathsf{explains}(\mathcal{M}^{\text{(-1)}}, t)$ with $\mathsf{explains}_{\mathcal{M}}^{\text{(-1)}}$; replace $\mathsf{explains}(\mathcal{M}^{\text{(-1)}} - \theta^{\text{(-1)}}, t)$ with $\mathsf{explains}_{\mathcal{M}-\theta}^{\text{(-1)}}$; and use similar replacements for $\mathsf{error}(\cdot)$, $\mathsf{covers}(\cdot)$, and $\mathsf{creates}(\cdot)$ (see Table 5.1):

$$\max\left\{\mathsf{explains}_{\mathcal{M}-\theta}^{\text{(-1)}}, \mathsf{covers}_\theta\right\} - \mathsf{explains}_{\mathcal{M}}^{\text{(-1)}} \tag{5.5}$$

For a given tuple t, a refinement may have a lower, equal, or greater value for $\mathsf{covers}(\cdot)$ than the original st tgd. Consider the nine cases, shown in Table 5.2, combining those three possibilities with whether (5.5) is negative, zero, or positive. On the top row, we know $\mathsf{covers}_\theta > \mathsf{covers}_\theta^{\text{(-1)}}$, so increasing (5.5) depends only on whether covers_θ surpasses $\mathsf{explains}_{\mathcal{M}}^{\text{(-1)}}$. On the bottom row, we know $\mathsf{covers}_\theta < \mathsf{covers}_\theta^{\text{(-1)}}$, so decreasing (5.5) only depends on whether $\mathsf{covers}_\theta^{\text{(-1)}}$ was the maximum. That is why

covers_θ appears in the top row and $\mathsf{covers}_\theta^{(-1)}$ appears in the bottom row. Case (b) on the top right shows that we only increase $\mathsf{explains}(\cdot)$ when $\mathsf{explains}_\mathcal{M}^{(-1)} < \mathsf{covers}_\theta$, which implies that $\mathsf{explains}_\mathcal{M}^{(-1)} < 1$. We want the score to be sensitive to that case, and we do not care about case (a). Although covers_θ varies across refinements, the $\mathsf{explains}_\mathcal{M}^{(-1)} < 1$ constraint is constant. Therefore, we can use it as a filter in our score over refinements, removing cases where $\mathsf{explains}_\mathcal{M}^{(-1)} = 1$. Case (c) shows that reducing $\mathsf{covers}(\cdot)$ on t can only lower $\mathsf{explains}(\cdot)$ if the old st tgd had the maximum value of $\mathsf{covers}(\cdot)$ for t over all st tgds in $\mathcal{M}^{(-1)}$. We want to disincentivize that case, and we do not care about case (d).

We can calculate the total change in the $\mathsf{explains}(\cdot)$ term as follows, using the two constraints from Table 5.2 (b) and (c) to remove cases we don't care about. We also remove the $\mathsf{explains}_\mathcal{M}^{(-1)}$ term of (5.5), as it is constant over all refinements.

$$\sum_{\substack{t \in J : \left(\mathsf{explains}_\mathcal{M}^{(-1)} < 1\right) \\ \vee \left(\mathsf{explains}_{\mathcal{M}-\theta}^{(-1)} < \mathsf{covers}_\theta^{(-1)}\right)}} \max\left\{\mathsf{explains}_{\mathcal{M}-\theta}^{(-1)}, \mathsf{covers}_\theta\right\} \tag{5.6}$$

Equation (5.6) is the total change in $\mathsf{explains}(\cdot)$ from replacing a st tgd with a refinement, but it is not suitable for scoring refinements. This is because our search may need multiple applications of refinement operators to increase $\mathsf{covers}(\cdot)$ above $\mathsf{explains}_{\mathcal{M}-\theta}^{(-1)}$, and the function $\max(\cdot)$ hides those incremental changes in $\mathsf{covers}(\cdot)$.

Removing the max provides more guidance over those steps:

$$\sum_{t \in J: \left(\mathsf{explains}^{(\text{-}1)}_{\mathcal{M}} < 1\right) \vee \left(\mathsf{explains}^{(\text{-}1)}_{\mathcal{M}-\theta} < \mathsf{covers}^{(\text{-}1)}_{\theta}\right)} \mathsf{covers}_{\theta} \tag{5.7}$$

5.3.3.2 Errors Boosting

The total change in the $\mathsf{error}(\cdot)$ term of the overall objective from replacing a st tgd with its refinement is the following:

$$\sum_{t \in K^{(\text{-}1)}_{\theta} \cup K_{\theta} - J} \left[\mathsf{creates}_{\theta} - \mathsf{creates}^{(\text{-}1)}_{\theta}\right] \tag{5.8}$$

The $\mathsf{creates}^{(\text{-}1)}_{\theta}$ terms of (5.8) are constant across refinements of $\theta^{(\text{-}1)}$, so we remove them:

$$\sum_{t \in K_{\theta} - J} \mathsf{creates}_{\theta} \tag{5.9}$$

The magnitude of change (Equation (5.8)) would be needed if we used s to compare refinements of different original st tgds, e.g., $\theta^{(\text{-}1)}_{a}$ and $\theta^{(\text{-}1)}_{b}$. However, we only use s to refine a single $\theta^{(\text{-}1)}$ at a time. Also notice that the sum over tuples t in Equation (5.9) is no longer defined over $K^{(\text{-}1)}_{\theta}$ because $\mathsf{creates}_{\theta}$ terms in Equation (5.9) will only be positive for tuples in K_{θ}.

5.3.3.3 Combined Score

We combine (5.7) and (5.9) as score $s^{(\text{-}1)}$ shown in Equation (5.4) and repeated here:

$$s(\theta^{(\text{-}1)}, \theta) = \sum_{\substack{t \in J : \left(\mathsf{explains}^{(\text{-}1)}_{\mathcal{M}} < 1\right) \\ \vee \left(\mathsf{explains}^{(\text{-}1)}_{\mathcal{M}-\theta} < \mathsf{covers}^{(\text{-}1)}_{\theta}\right)}} [\mathsf{covers}(\theta, t)] - \sum_{t \in K_\theta - J} [\mathsf{creates}(\theta, t)]$$

The $\mathsf{covers}(\cdot)$ term of Equation (5.4) is based on Equation (5.7). The $\mathsf{creates}(\cdot)$ term, based on Equation (5.9), is negated so that higher values for s are better.

The constraints over target tuples in the sums of (5.4) are equivalent to weights of zero or one on tuples, used in a version of the score using weights. Set $\vec{w}$ of weights is used in the $\mathsf{covers}(\cdot)$ term:

$$s^{(\text{-}1)}(\theta) = \sum_{t \in J} [w_t \times \mathsf{covers}(\theta, t)] - \sum_{t \in K_\theta - J} [\mathsf{creates}(\theta, t)] \tag{5.10}$$

We illustrate weights $\vec{w}$ in Figure 5.4, which shows the following sets of target tuples:

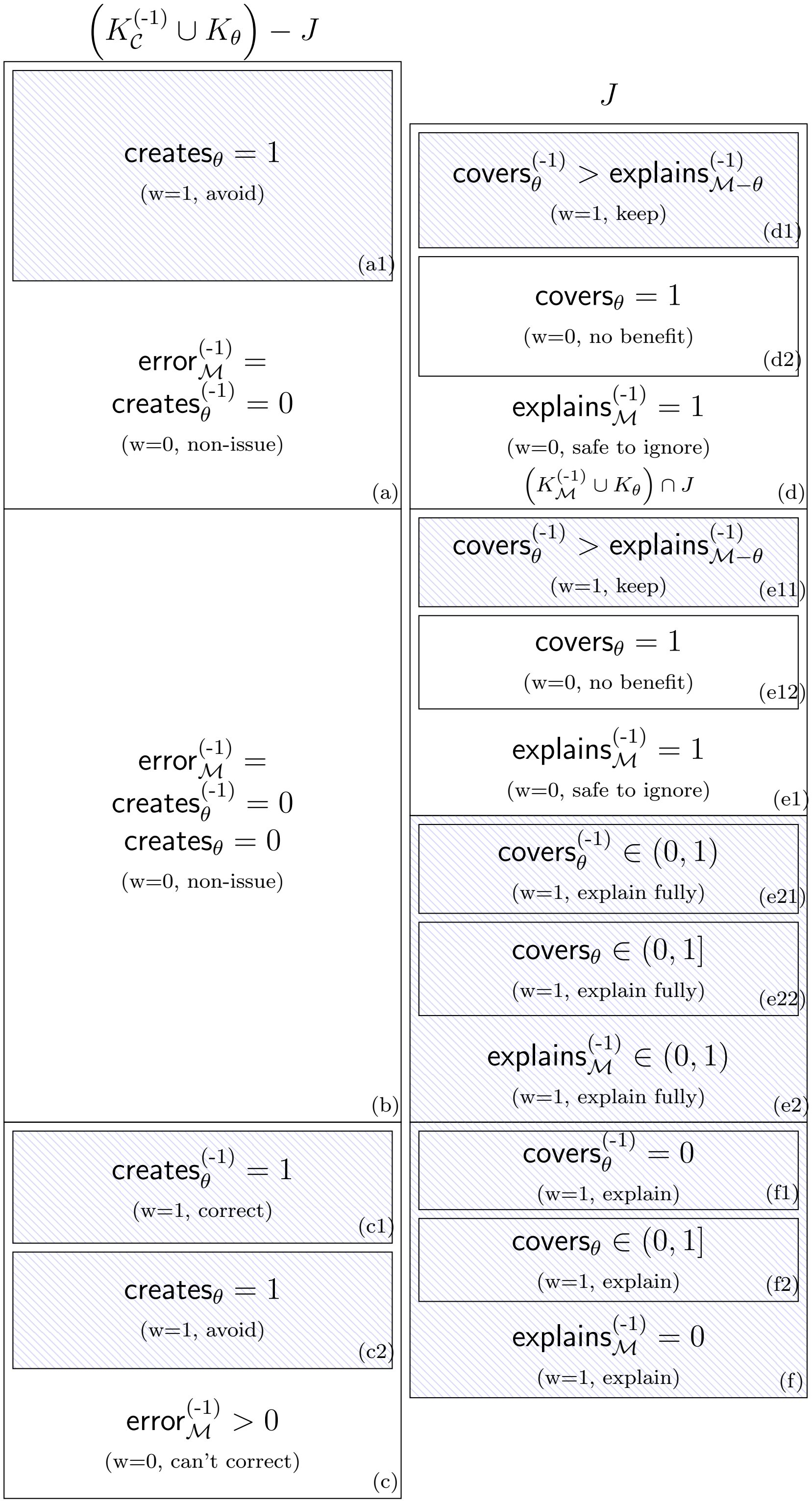

$\left(K_{\mathcal{C}}^{(-1)} \cup K_\theta\right) - J$
$\text{creates}_\theta = 1$
(w=1, avoid)
(a1)
$\text{error}_{\mathcal{M}}^{(-1)} =$
$\text{creates}_\theta^{(-1)} = 0$
(w=0, non-issue)
(a)
$\text{error}_{\mathcal{M}}^{(-1)} =$
$\text{creates}_\theta^{(-1)} = 0$
$\text{creates}_\theta = 0$
(w=0, non-issue)
(b)
$\text{creates}_\theta^{(-1)} = 1$
(w=1, correct)
(c1)
$\text{creates}_\theta = 1$
(w=1, avoid)
(c2)
$\text{error}_{\mathcal{M}}^{(-1)} > 0$
(w=0, can't correct)
(c)
J
$\text{covers}_\theta^{(-1)} > \text{explains}_{\mathcal{M}-\theta}^{(-1)}$
(w=1, keep)
(d1)
$\text{covers}_\theta = 1$
(w=0, no benefit)
(d2)
$\text{explains}_{\mathcal{M}}^{(-1)} = 1$
(w=0, safe to ignore)
$\left(K_{\mathcal{M}}^{(-1)} \cup K_\theta\right) \cap J$
(d)
$\text{covers}_\theta^{(-1)} > \text{explains}_{\mathcal{M}-\theta}^{(-1)}$
(w=1, keep)
(e11)
$\text{covers}_\theta = 1$
(w=0, no benefit)
(e12)
$\text{explains}_{\mathcal{M}}^{(-1)} = 1$
(w=0, safe to ignore)
(e1)
$\text{covers}_\theta^{(-1)} \in (0, 1)$
(w=1, explain fully)
(e21)
$\text{covers}_\theta \in (0, 1]$
(w=1, explain fully)
(e22)
$\text{explains}_{\mathcal{M}}^{(-1)} \in (0, 1)$
(w=1, explain fully)
(e2)
$\text{covers}_\theta^{(-1)} = 0$
(w=1, explain)
(f1)
$\text{covers}_\theta \in (0, 1]$
(w=1, explain)
(f2)
$\text{explains}_{\mathcal{M}}^{(-1)} = 0$
(w=1, explain)
(f)

Figure 5.4: Illustration of Equation (5.10). Tuples in shaded sections have weight one; all others have zero weight.

- J from the example
- $K_{\mathcal{C}}^{\text{(-1)}}$ from existing candidates
- K_θ from the current refinement

The diagram then subdivides each set into those associated with the original st tgd $\theta^{\text{(-1)}}$, or a refinement θ. The shaded sections of the figure represent tuples with weight one, and all others have zero weight in the score. The $\mathsf{covers}(\cdot)$ term in (5.4) prefers refinements that explain more tuples. It is summed over the union of two sets of tuples in J: (1) tuples not fully explained by $\mathcal{M}^{\text{(-1)}}$ (Figure 5.4 (e2) and (f)), and (2) tuples explained best by $\theta^{\text{(-1)}}$ (Figure 5.4 (d1) and (e11)). Naturally, we want refinements to explain the first set fully and leave the second explained, i.e., avoiding refinements that leave already-explained tuples unexplained. We implement those preferences by giving both sets of tuples weight one. The coverage of other tuples in J is unaffected when refining $\theta^{\text{(-1)}}$, as they are already explained by other st tgds (Figure 5.4 (d), (d2), (e1) and (e12)). So, we exclude them from the score, i.e., they receive a weight of zero. Notice that, although we selected $\theta^{\text{(-1)}}$ based on relation T, our score for refinements is based on *all relations of* J, rather than $J(T)$. This is to ensure we take into account the multi-relational structure possible with shared nulls across relations. By leaving already-explained tuples out of the score, we reduce the number of homomorphism checks needed to calculate $\mathsf{covers}(\cdot)$. More importantly, we re-weight the example for boosting.

Consider an alternate score having no penalty for a refinement θ reducing $\mathsf{covers}(\cdot)$ for some tuple t, compared to $\theta^{\text{(-1)}}$. We never remove candidates from $\mathcal{C}$, so MappingSelection could still explain t by selecting both θ and $\theta^{(\text{-}1)}$ in $\mathcal{M}^{(m)}$. The errors made by the st tgds, however, would accumulate. For example, if both st tgds make similar errors, they each contribute to the total errors. If θ corrected an error made by $\theta^{(\text{-}1)}$, selecting both negates that improvement, as $\theta^{(\text{-}1)}$ will still generate the error. The $\mathsf{creates}(\cdot)$ term in (5.4) prefers refinements that correct errors made by $\theta^{(\text{-}1)}$ (Figure 5.4(c1)) or θ (Figure 5.4(a1, c2)).

5.3.4 Step 2.2: Refine ST TGDs

We next use a local search subroutine to refine each st tgd in $\Theta^{(\text{-}1)}$, approximating the objective in Equation (5.3). Function Refine($\theta^{(\text{-}1)}, \mathcal{C}^{(\text{-}1)}, s^{(\text{-}1)}, \boldsymbol{\rho}, B, U$) shown in Algorithm 2, uses refinement operators $\boldsymbol{\rho}$ to generate new st tgd θ. Algorithm 1 uses Refine($\cdot$) on lines 9 and 10 to expand the set of candidates. Function Refine($\cdot$) uses a local search, initialized with $\theta^{(\text{-}1)}$. To include a new refinement of a st tgd, it must have at least as high a score as the original st tgd. The refinement must also be well formed; specifically, we check that st tgds have at least one logical variable that is in both the body and the head. We describe the set of refinement operators in Section 5.4.

5.3.5 Step 3: Update Mapping

Finally, we select a new, optimal subset $\mathcal{M}^{(m)} \subseteq \mathcal{C}^{(m)}$ (line 12) as the current mapping. The flaws of this updated mapping will drive the search for a better mapping in the following search stage.

```
Algorithm 2: Refine
Data: $\theta^{(-1)}$, $\mathcal{C}^{(-1)}$, $s^{(-1)}$, $\boldsymbol{\rho}$, $U$
Result: $\theta$
$\theta^{(-1)}$.score ← $\mathrm{s}^{(-1)}(\theta^{(-1)})$;
for $u$ from 1 to $U$ do
    $\theta^{(u)} \leftarrow \theta^{(-1)}$;
    $\mathbf{R} \leftarrow \{\}$;
    for $\rho \in \boldsymbol{\rho}$ do
        for $r \in \rho(\theta^{(u)})$ do
            add $r$ to $\mathbf{R}$ if $r$ is well formed and not previously seen;
        end
    end
    for $r \in \mathbf{R}$ do
        $r$.score ← $\mathrm{s}^{(-1)}(r)$;
        if $r$.score $\geq \theta^{(u)}$.score then
            $\theta^{(u)} \leftarrow r$;
        end
    end
end
return $\theta^{(U)}$;
```

5.4 Refinement Operators

In this section, we describe the refinement operators that we use in our search approach (Section 5.3.4). A refinement operator is a function that takes as input a st tgd of the form $\forall \vec{x}, \vec{z}.\ \phi(\vec{x}, \vec{z}) \rightarrow \exists \vec{y}\ \psi(\vec{y}, \vec{z})$, where $\phi(\vec{x}, \vec{z})$ and $\psi(\vec{y}, \vec{z})$ are conjunctions of atoms. Recall that notation $\lambda\{u/v\}$ denotes the formula derived from λ by substituting all occurrences of u with v. For example, $\theta(\vec{x}, \vec{y}, \vec{z})\ \{y_j/x_k\}$ replaces every occurrence of variable y_j in st tgd θ with variable x_k. As a result, non-exchanged variable x_k appears in both the body and head, and becomes a member of the exchanged set $\vec{z}$. We superficially post-process each refinement this way to reflect that variables can move between sets $\vec{x}$, $\vec{y}$, and $\vec{z}$ as a result of refinement. In this section, we first motivate our choice of refinement operators by describing informally the types of flaws that they need to correct. We then define each operator formally.

5.4.1 Types of Mapping Flaws

Consider an extremely simple st tgd $\theta' = \forall \vec{x}.\ \phi'(\vec{x}) \rightarrow \exists \vec{y}\ \psi'(\vec{y})$ having a unique variable in every argument of every atom of ϕ', a unique variable in every argument of every atom of ψ', and no exchanged variables. The following flaws may arise when using θ' in a mapping:

1. Every argument of ϕ' is assigned a variable, allowing it to ground on every atom of its relations in I. To avoid errors, it may be necessary to constrain one or more arguments in ϕ' to have the same value, i.e., to merge their variables.

2. A special case of flaw (1) is when it is necessary to add an atom to ϕ', enabling useful constraints on groundings and exchanging additional data.

3. θ' has no exchanged variables and all variables in ψ' are existential. Although this enables homomorphisms with the potential of explaining tuples of relations in J, it only produces nulls in those tuples, so $\mathsf{covers}(\cdot)$ will be low — in fact, with unique existential variables, it will be zero. One or more variables in ϕ' may need to be merged with existential variables in ψ' to become exchanged variables.

4. Similar to flaw (2), ψ' may lack atoms that could be used to explain more tuples. It may be necessary to add one or more atoms, joining them to ψ' by

merging variables.

5. All existential variables in ψ' are unique, i.e., a different one in every argument, but structure from shared nulls in the target instance is important, and rewarded in our objective (see Equation (5.1)). One or more existential variables may need to be merged to create that structure.

6. The head of ψ' may be correct, but its body is flawed, e.g., using the wrong source relation. Instead of gradually changing the body, or building a new st tgd from scratch, it could be faster to replace the body entirely.

The st tgds we construct from scratch in Section 5.3.2 have similar flaws as θ', and the same may be true of st tgds in $\mathcal{C}^{(0)}$. These flaws motivate our use of operators in three categories: *substitution* operators, *conjunction* operators, and *antecedent* operators. We describe each category in the following three sections.

5.4.2 Substitution Operators

Substitution operators are designed to address flaws 1, 3, and 5 from Section 5.4.1. Each substitution operator works over some pair of sets of variables, where each set is one of the three $(\vec{x}, \vec{y}, \vec{z})$ in some st tgd. The operator works by substituting members of one set for members of the other set. We list the substitution operators we use in Figure 5.5. For example, $\rho_{\mathtt{xx}\phi}$ creates a refinement for every pair of non-exchanged variables, where those variables are merged (see flaw (1)). $\rho_{\mathsf{zz}\theta}$ is serves a similar purpose to $\rho_{\mathtt{xx}\phi}$, but merges variables that are exchanged. Both of these operators limit the groundings of refinements on I. None introduce new errors

$$\begin{aligned}
\rho_{\mathtt{xx}\phi}(\theta) &= \{\theta(\vec{x}, \vec{y}, \vec{z})\, \{x_j/x_k\} \text{ with } x_j \in \vec{x}, x_k \in \vec{x}, x_j \neq x_k\} \\
\rho_{\mathtt{xy}\phi}(\theta) &= \{\theta(\vec{x}, \vec{y}, \vec{z})\, \{x_j/y_k\} \text{ with } x_j \in \vec{x}, y_k \in \vec{y}\} \\
\rho_{\mathtt{xz}\phi}(\theta) &= \{\theta(\vec{x}, \vec{y}, \vec{z})\, \{x_j/z_k\} \text{ with } x_j \in \vec{x}, z_k \in \vec{z}\} \\
\rho_{\mathtt{yx}\psi}(\theta) &= \{\theta(\vec{x}, \vec{y}, \vec{z})\, \{y_j/x_k\} \text{ with } y_j \in \vec{y}, x_k \in \vec{x}\} \\
\rho_{\mathtt{yy}\psi}(\theta) &= \{\theta(\vec{x}, \vec{y}, \vec{z})\, \{y_j/y_k\} \text{ with } y_j \in \vec{y}, y_k \in \vec{y}, y_j \neq y_k\} \\
\rho_{\mathtt{yz}\psi}(\theta) &= \{\theta(\vec{x}, \vec{y}, \vec{z})\, \{y_j/z_k\} \text{ with } y_j \in \vec{y}, z_k \in \vec{z}\} \\
\rho_{\mathtt{zx}\phi}(\theta) &= \{\phi(\vec{x}, \vec{z})\, \{z_j/x_k\} \to \exists\vec{y}.\ \psi(\vec{y}, \vec{z}) \text{ with } z_j \in \vec{z}, x_k \in \vec{x}\} \\
\rho_{\mathtt{zx}\psi}(\theta) &= \{\phi(\vec{x}, \vec{z}) \to \exists\vec{y}.\ \psi(\vec{y}, \vec{z})\, \{z_j/x_k\} \text{ with } z_j \in \vec{z}, x_k \in \vec{x}\} \\
\rho_{\mathtt{zy}\psi}(\theta) &= \{\phi(\vec{x}, \vec{z}) \to \exists\vec{y}.\ \psi(\vec{y}, \vec{z})\, \{z_j/y_k\} \text{ with } z_j \in \vec{z}, y_k \in \vec{y}\} \\
\rho_{\mathtt{zz}\phi}(\theta) &= \{\phi(\vec{x}, \vec{z})\, \{z_j/z_k\} \to \exists\vec{y}.\ \psi(\vec{y}, \vec{z}) \text{ with } z_j \in \vec{z}, z_k \in \vec{z}, z_j \neq z_k\} \\
\rho_{\mathtt{zz}\psi}(\theta) &= \{\phi(\vec{x}, \vec{z}) \to \exists\vec{y}.\ \psi(\vec{y}, \vec{z})\, \{z_j/z_k\} \text{ with } z_j \in \vec{z}, z_k \in \vec{z}, z_j \neq z_k\} \\
\rho_{\mathtt{zz}\theta}(\theta) &= \{\theta(\vec{x}, \vec{y}, \vec{z})\, \{z_j/z_k\} \text{ with } z_j \in \vec{z}, z_k \in \vec{z}, z_j \neq z_k\}
\end{aligned}$$

Figure 5.5: Substitution operators over st tgd $\theta = \forall\vec{x}, \vec{z}.\ \phi(\vec{x}, \vec{z}) \to \exists\vec{y}\ \psi(\vec{y}, \vec{z})$.

or increase coverage of tuples in J. They can correct errors, and they can completely remove coverage of tuples; score $s^{(\text{-}1)}$ penalizes the latter.

5.4.3 Conjunction Operators

Conjunction operators are designed to address flaws 2 and 4 from Section 5.4.1. We list the conjunction operators in Figure 5.6. For example, $\rho_{\mathtt{wy}\psi}$ creates refinements having added head joins for every combination of target relation and existential variable (see flaw (4)). Two operators – $\rho_{\text{--}\phi}$ and $\rho_{\text{--}\psi}$ – are designed for the complementary purpose of removing atoms from the body and head, respectively.

5.4.4 Antecedent Operators

Antecedent operators are designed to address flaw 6 from Section 5.4.1, which requires that we entirely replace the antecedent (body) of the st tgd. The two

$$
\begin{aligned}
\rho_{\texttt{wx}\phi}(\theta) &= \{\phi(\vec{x},\vec{z}) \wedge R(\vec{w})\,\{w_j/x_k\} \rightarrow \exists\vec{y}.\ \psi(\vec{y},\vec{z}) \text{ with } R \in \mathbf{S}, w_j \in \vec{w}, x_k \in \vec{x}\} \\
\rho_{\texttt{wy}\psi}(\theta) &= \{\phi(\vec{x},\vec{z}) \rightarrow \exists\vec{y}.\ \psi(\vec{y},\vec{z}) \wedge R(\vec{w})\,\{w_j/y_k\} \text{ with } R \in \mathbf{T}, w_j \in \vec{w}, y_k \in \vec{y}\} \\
\rho_{\texttt{wz}\phi}(\theta) &= \{\phi(\vec{x},\vec{z}) \wedge R(\vec{w})\,\{w_j/z_k\} \rightarrow \exists\vec{y}.\ \psi(\vec{y},\vec{z}) \text{ with } R \in \mathbf{S}, w_j \in \vec{w}, z_k \in \vec{z}\} \\
\rho_{\texttt{wz}\psi}(\theta) &= \{\phi(\vec{x},\vec{z}) \rightarrow \exists\vec{y}.\ \psi(\vec{y},\vec{z}) \wedge R(\vec{w})\,\{w_j/z_k\} \text{ with } R \in \mathbf{T}, w_j \in \vec{w}, z_k \in \vec{z}\} \\
\rho_{\texttt{--}\phi}(\theta) &= \left\{ \bigwedge_{i\in[1,|\phi|], i\neq j} [\phi_i(\vec{x},\vec{z})] \rightarrow \exists\vec{y}.\ \psi(\vec{y},\vec{z}) \text{ with } j \in [1,|\phi|] \right\} \\
\rho_{\texttt{--}\psi}(\theta) &= \left\{ \phi(\vec{x},\vec{z}) \rightarrow \exists\vec{y}.\ \bigwedge_{i\in[1,|\psi|], i\neq j} [\psi_i(\vec{x},\vec{z})] \text{ with } j \in [1,|\psi|] \right\}
\end{aligned}
$$

Figure 5.6: Conjunction operators over st tgd $\theta = \forall\vec{x},\vec{z}.\ \phi(\vec{x},\vec{z}) \rightarrow \exists\vec{y}\ \psi(\vec{y},\vec{z})$.

antecedent rules are as follows:

$$\rho_{\texttt{-y}\phi}(\theta) = \{R(\vec{w})\,\{w_j/y_k\} \rightarrow \exists\vec{y}.\ \psi(\vec{y},\vec{z}) \text{ with } R \in \mathbf{S}, w_j \in \vec{w}, y_k \in \vec{y}\}$$

$$\rho_{\texttt{-z}\phi}(\theta) = \{R(\vec{w})\,\{w_j/z_k\} \rightarrow \exists\vec{y}.\ \psi(\vec{y},\vec{z}) \text{ with } R \in \mathbf{S}, w_j \in \vec{w}, z_k \in \vec{z}\}$$

For example, $\rho_{\texttt{-y}\phi}$ creates refinements for every combination of source relation and existential variable. An important use of the antecedent operators is with the st tgds we construct from scratch in Section 5.3.2. Those st tgds are missing a body entirely and the antecedent operators create refinements that are well-formed.

5.5 Baseline Algorithms

Search function $\mathcal{S}_{\texttt{sib}}$ (Algorithm 1) uses the objective in Equation (5.1) as a basis for both selection and iterative refinement. Our goal will be to evaluate the

Algorithm 3: Function $\mathcal{S}_{\texttt{ib}}$ lacks selection.

```
   Data:
   Refinement operator set: $\boldsymbol{\rho}$
   Search stages: $M$
   Refinement iterations: $U$
   Refinement score: $s$
   Result: $\mathcal{M}$
1  $\mathcal{M}^{(0)} = \{\}$;
2  for $m$ from 1 to $M$ do
3      $\mathcal{M}^{(m)} \leftarrow \mathcal{M}^{(\text{-}1)}$;
4      Select relation $T \in \mathbf{T}$;
5      Stop if total for $T$ in Equation (5.2) is zero;
6      $\theta^{(\text{-}1)}$ is a new st tgd with $T$;
7      Update weights used by score function $s^{(\text{-}1)} = s(\theta^{(\text{-}1)}, \theta)$;
8      $\theta \leftarrow \text{Refine}(\theta^{(\text{-}1)}, \mathcal{M}^{(\text{-}1)}, s^{(\text{-}1)}, \boldsymbol{\rho}, U)$;
9      $\mathcal{M}^{(m)} \leftarrow \mathcal{M}^{(m)} \cup \{\theta\}$;
10 end
11 return $\mathcal{M}^{(M)}$;
```

effectiveness of those two contributions. Therefore, we will compare $\mathcal{S}_{\texttt{sib}}$ with two baseline functions: one lacks selection and the other lacks iteration.

5.5.1 Function $\mathcal{S}_{\texttt{ib}}$

Search function $\mathcal{S}_{\texttt{ib}}$ lacks a selection step and is representative of conventional single-relation learning approaches. We list $\mathcal{S}_{\texttt{ib}}$ in Algorithm 3. Like $\mathcal{S}_{\texttt{sib}}$, $\mathcal{S}_{\texttt{ib}}$ uses boosting, but it does not use MappingSelection($\cdot$). That is, in each stage it selects a relation (line 4), it starts a single st tgd from scratch for that relation, and updates weights (line 7) based on the latest mapping. Each stage, it selects a single best refinement (line 8) and adds it to the mapping (line 9). In contrast, all members of $\mathcal{M}^{(m)}$ may change in each stage of $\mathcal{S}_{\texttt{sib}}$ during its selection step (line 12).

Algorithm 4: Search function $\mathcal{S}_{\mathtt{s}}$ uses selection

```
Data:
Initial candidate set: $\mathcal{C}$
Result: $\mathcal{M}$
return MappingSelection($\mathcal{C}$);
```

5.5.2 Function $\mathcal{S}_{\mathtt{s}}$

Function $\mathcal{S}_{\mathtt{s}}$, shown in Algorithm 4, uses the selection approach only. It uses no refinement operators; instead it assumes input $\mathcal{C}$ contains sufficient candidates. We use CMD for selection (Chapter 3).

5.5.3 Search Inputs and Parameters

The three search functions use slightly different inputs and parameters, so we now explain how we apply each algorithm to the same mapping problem inputs, and how we vary the algorithm parameters in the evaluation (Section 5.7).

Functions $\mathcal{S}_{\mathtt{sib}}$ and $\mathcal{S}_{\mathtt{s}}$ both use selection, so use a set $\mathcal{C}^{(0)}$ of candidates as input. Function $\mathcal{S}_{\mathtt{ib}}$ lacks a selection step and instead starts from an empty mapping, adding a new rule for the target relation selected in each stage. Two of the functions ($\mathcal{S}_{\mathtt{sib}}$ and $\mathcal{S}_{\mathtt{ib}}$) are iterative and use a set $\boldsymbol{\rho}$ of refinement operators; the exception is $\mathcal{S}_{\mathtt{s}}$. For both, we use the full set of operators defined in Section 5.4. The two functions using boosting ($\mathcal{S}_{\mathtt{sib}}$ and $\mathcal{S}_{\mathtt{ib}}$) both use the same weighted refinement score function s defined in Equation (5.4).

Both iterative functions stop after the same maximum number M of iterations, or when the algorithm finds no remaining flaws at the beginning of a stage. Both have both inner and outer iteration loops; they use a second parameter U for the local search subroutine that runs the inner loop. Except where noted, to evaluate an iterative function, we run it multiple times on the same problem while varying the function parameter settings. We use one setting for each combination of M and U, as follows: We vary M from one to five in steps of two. We vary U from one to seven in steps of two.

5.6 Scenario Generation

In Chapter 3, we use iBench [43, 76] – a system for generating diverse and realistic integration scenarios – to evaluate the CMD mapping approach ($\mathcal{S}_{\mathbf{s}}$). In this section, we go further and define a novel set of generated integration scenarios with parameters specifically for controlling the search problem difficulty. Each problem comprises the following:

- A pair of schemas and a data example: $(\mathbf{S}, \mathbf{T}, I, J)$
- One gold mapping: $\mathcal{M}_G$
- A set of st tgds: $\mathcal{C} = \mathcal{C}^{(0)}$

All problems have the following parameters in common, except where noted:

- Source relations $\mathbf{S} = \{s, s'\}$, each with arity k_s.
- Target relations $\mathbf{T} = \{t, t'\}$, each with arity k_t.
- Arguments of s are indexed left-to-right by i_s. Those of s' are indexed by ${i_s}'$.

- Arguments of t are indexed left-to-right by i_t. Those of t' are indexed by ${i_t}'$.
- Universally-quantified variables $\{X_1, \ldots, X_{k_s}\}$ for s, and $\{X'_1, \ldots, X'_{k_s}\}$ for s'.
- Existentially-quantified variables $\{Y_1, \ldots, Y_{k_t}\}$ for t, and $\{Y'_1, \ldots, Y'_{k_t}\}$ for t'.
- Source instance I contains $m_s = 50$ tuples in each source relation.
- Target instance J contains $m_t = 50$ tuples in each target relation.
- Constants in I are mutually exclusive across arguments of all source relations.
- Constants in J are mutually exclusive across arguments of all target relations.

The arities of relations have a direct impact on the complexity of the search problem. Except where noted, we generate scenarios by varying arity (k_s or k_t or both) from one to five in steps of two. We group our generated problems into four widely-used classes of mappings that vary by the subset of the st tgd language they use. In the next four sections, we define the problems in each of the four classes.

5.6.1 Single-Head Local-as-View

Single-Head Local-as-View (SH-LAV) mappings are restricted to a single source relation $\mathbf{S} = \{s\}$ and a single target relation $\mathbf{T} = \{t\}$. Despite these restrictions, the space of possible mappings is large because logical variables can appear in any arrangement in the body of the st tgd, in the head, or both.

Mapping functions should handle flawed input st tgds that make errors. One way that errors vary is the location of incorrect arguments. In the following problem, we set $k_s = 1$ and vary which argument is incorrect in errors made by st tgds in $\mathcal{C}$:

$$
\begin{aligned}
\mathcal{M}_G &= \{s(X_1) \rightarrow t(X_1, Y_2, Y_3, \ldots, Y_{k_t-1}, Y_{k_t})\} \\
\mathcal{C} &= \{s(X_1) \rightarrow t(Y_1, \ldots, Y_{i_t-1}, X_1, Y_{i_t+1}, \ldots, Y_{k_t}) \\
&\qquad \text{with } i_t \in [1, k_t]\}
\end{aligned}
\tag{5.11}
$$

Problem 5.11 has $k_t - 1$ st tgds that make errors; we refer to these as *bad* st tgds. Each bad st tgd has one bad argument; the other arguments have existentially quantified variables.

The second problem set Problem 5.12 has the same gold mapping and a set of candidates that vary by the number of copies of variable X_1 in the head, which varies the number (out of k_t) of incorrect arguments:

$$
\begin{aligned}
\mathcal{C} &= \{s(X_1) \rightarrow t(X_1, X_1, \ldots, X_1, Y_{i_t+1}, \ldots, Y_{k_t}) \\
&\qquad \text{with } i_t \in [1, k_t]\}
\end{aligned}
\tag{5.12}
$$

Problem 5.12 also has $k_t - 1$ bad st tgds; each has $i_t - 1$ bad arguments, one correct argument, and the remainder are existential.

Mapping functions should also handle flawed st tgds that do not fully explain tuples. In the following problem, we set $k_s = k_t$ and vary the number of arguments (out of k_t) of tuples of J that are explained by st tgds in $\mathcal{C}$:

$$
\begin{aligned}
\mathcal{M}_G = \{ & s(X_1, X_2, \ldots, X_{k_s}) \rightarrow \\
& t(X_1, X_2, \ldots, X_{k_s})\} \\
\mathcal{C} = \{ & s(X_1, X_2, \ldots, X_{k_s}) \rightarrow \\
& t(X_1, X_2, \ldots, X_{i_t}, Y_{i_t+1}, \ldots, Y_{k_t}) \\
& \text{with } i_t \in [1, j]\}
\end{aligned} \tag{5.13}
$$

Parameter $1 \leq j \leq k_t$ controls the maximum number of unexplained arguments in candidates. If $j = k_t$, Problem 5.13 has $k_t - 1$ bad st tgds that vary by the number of explained arguments. If $j < k_t$, all st tgd are only partly explained. We generate scenarios by varying j from three to k_t in steps of two.

Mapping functions should be able to discover selection conditions with equalities. In the following problem, we generate the source instance I so that the simple candidate st tgd causes errors in the target. Specifically, $m_t = 75$ and $m_s = 100$, where 25 of the source tuples should not be transferred to the target. To avoid the errors, we have to discover a mapping with a selection condition setting all but one of the k_s source terms equal.

$$
\begin{aligned}
\mathcal{M}_G &= \{s(X_1, X_2, X_2, \ldots, X_2) \rightarrow t(X_1)\} \\
\mathcal{C} &= \{s(X_1, X_2, X_3, \ldots, X_{k_s}) \rightarrow t(X_1)
\end{aligned} \tag{5.14}
$$

Mapping functions should be able to discover shared nulls, i.e., constraints on invented values. The following problem is similar to Problem 5.14 because we need to learn equality conditions over variables, except we now need to learn them over terms in the target, not the source.

$$
\begin{aligned}
\mathcal{M}_G &= \{s(X_1) \rightarrow t(X_1, Y_2, Y_2, \ldots, Y_2)\} \\
\mathcal{C} &= \{s(X_1) \rightarrow t(X_1, Y_2, Y_3, \ldots, Y_{k_t}) \qquad (5.15)
\end{aligned}
$$

The simple candidate st tgd is able to cover all the tuples in J, as homomorphisms will exist from each of its $k_t - 1$ unique existential variables to tuples of J. However, the revised covers function (Section 5.2.1.1) prefers the constraints provided by the correct st tgd which replaces all existentials with a single existential, and therefore a single labeled null.

5.6.2 Local-as-View

Local-as-View (LAV) mappings are restricted to a single source relation $\mathbf{S} = \{s\}$ and can have one or more target relations. Mapping functions should be able to learn the target relation joins that are possible with LAV mappings. In the following problem, we must learn a correct join in order to explain tuples of the additional relation t'.

$$\begin{aligned}\mathcal{M}_G = \{&s(X_1, X_2, \ldots, X_{k_s}) \rightarrow \\ &\quad t(Y_1, X_2, \ldots X_{k_s/2}) \wedge t'(Y_1, X_{k_t/2+2}, \ldots, X_{k_s})\} \\ \mathcal{C} = \{&s(X_1, X_2, \ldots X_{k_s}) \rightarrow t(Y_1, X_2, Y_3, \ldots, Y_{k_t})\}\end{aligned} \tag{5.16}$$

We set $k_t = k_s/2$, which means that learning the correct join can double the coverage of the simple candidate st tgd.

5.6.3 Global-as-View

Global-as-View (GAV) mappings are restricted to a single target relation $\mathbf{T} = \{t\}$. Mapping functions should be able to learn the source relation joins that are possible with GAV mappings. The following problem is similar to Problem 5.16 because we need to discover a correct join, except the join is over source relations, not target relations.

$$\begin{aligned}\mathcal{M}_G = \{&s(X_1, X_2, \ldots X_{k_s}) \wedge s'(X_1, X'_2, \ldots, X'_{k_s}) \rightarrow \\ &\quad t(X_1, X_2, X_3, \ldots, X_{k_s}, X'_2, X'_3, \ldots, X'_{k_s})\} \\ \mathcal{C} = \{&s(X_1, X_2, \ldots X_{k_s}) \rightarrow t(X_1, Y_2, Y_3, \ldots, Y_{k_t})\}\end{aligned} \tag{5.17}$$

In this problem, the simple candidate st tgd covers all the tuples in J; however, it leaves half of the arguments filled with nulls. We set $k_t = 2k_s - 1$, which means that learning the correct join can double the number of arguments that are filled with constants instead of nulls.

5.6.4 Global-Local-as-View

Global-Local-as-View (GLAV) mappings can have multiple source and target relations. Mapping functions should be able to learn the source and target joins that are possible with GLAV mappings. The following problem combines the challenges of problems 5.16 and 5.17.

$$\begin{aligned}\mathcal{M}_G = \{s(X_1, X_2, \ldots X_{k_s}) \wedge s'(X_1, X_2', \ldots, X_{k_s}') \rightarrow \\ t(Y_1, X_2, \ldots, X_{k_s}) \wedge t'(Y_1, X_2', \ldots, X_{k_s}')\} \\ \mathcal{C} = \{s(X_1, X_2, \ldots X_{k_s}) \rightarrow t(Y_1, X_2, Y_3, \ldots, Y_{k_t})\}\end{aligned} \tag{5.18}$$

In this problem $k_t = k_s$; explaining all target relations fully requires learning the correct join over the source, the correct join over the target, and learning the correct exchanged variables from all source relations to the correct target relations. Therefore, this problem is the most complex in the problem set.

5.7 Evaluation

In this section, we empirically evaluate the proposed mapping search approach, comparing it to several baseline algorithms with respect to mapping quality and scalability. We use 16 generated mapping problems and 28 algorithm parameter configurations, resulting in 448 total measures of mapping quality and running time. Additionally we demonstrate the mapping search approach on two real data problems. We ran our experiments on an Intel Xeon with 24 x 2.67GHz CPU and 128GB RAM.

5.7.1 Mapping Quality

We use the problems in Section 5.6 to evaluate the ability of the search and baseline algorithms to discover high quality mappings. As we did in Chapter 3, to measure the correctness of the mappings selected by our approach, we used the IQ-Meter [61] evaluation measure. IQ-Meter compares the results $J_{\mathcal{M}}$ of our learned mapping to the solution J_G of the gold standard mapping. IQ-Meter measures the ability of a mapping to generate correct tuples as well as correct relational structures via shared nulls, so it is appropriate as an evaluation measure on the mapping problem. It calculates recall and precision of tuples and recall and precision of joins. It also combines these four measures into a combined measure $\mathsf{score}(J_{\mathcal{M}}, J_G) \in [0, 1]$, where 1.0 is the best possible score for a mapping. We also measure the consistency of mapping quality from each algorithm as the standard deviation of the quality score.

In Figures 5.7 through 5.12 we plot the mean and standard deviation of quality scores for the three algorithms on all the generated scenarios for all problems over all scenario parameter settings. Our proposed search algorithm $\mathcal{S}_{\mathtt{sib}}$ has the highest mean quality over-all; it also has the highest quality for the tuple and join components

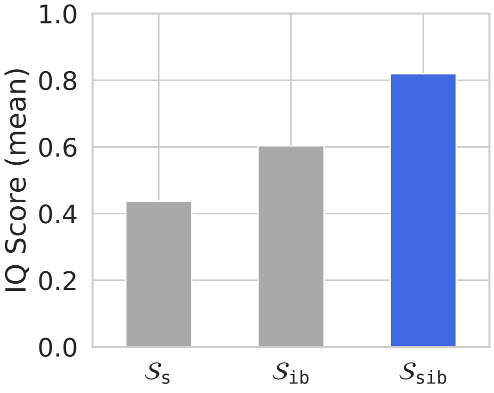


Figure 5.7: Mapping quality

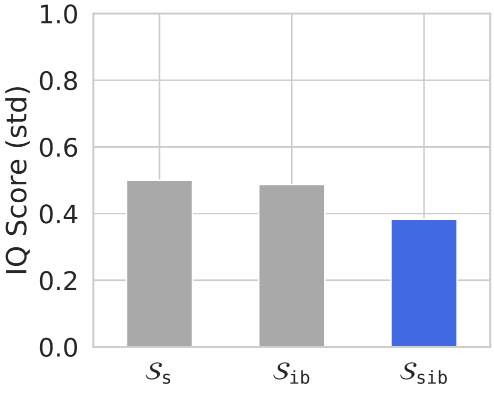


Figure 5.8: Mapping quality variation

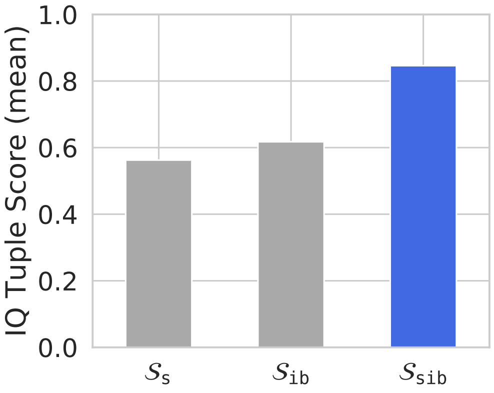


Figure 5.9: Tuple quality

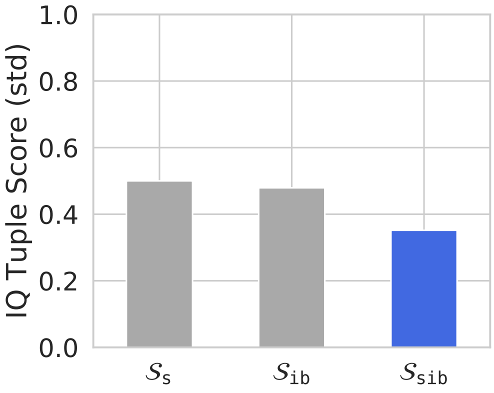


Figure 5.10: Tuple quality variation

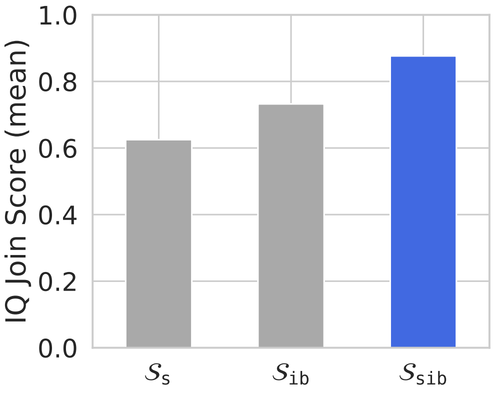


Figure 5.11: Join quality

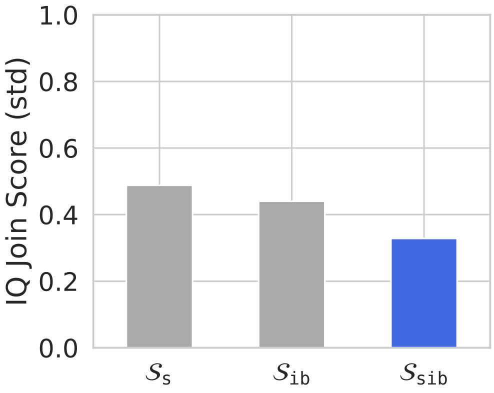


Figure 5.12: Join quality variation

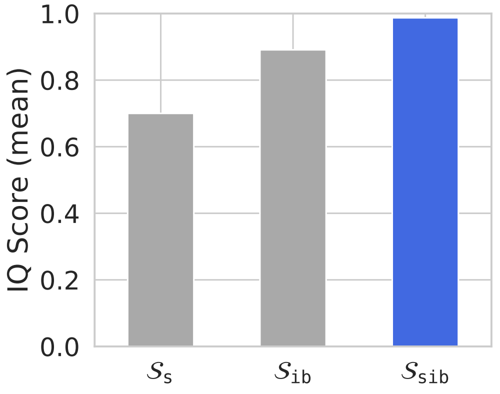


Figure 5.13: SHLAV mapping quality

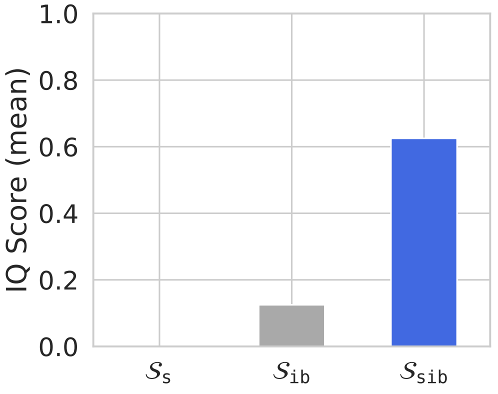


Figure 5.14: LAV mapping quality

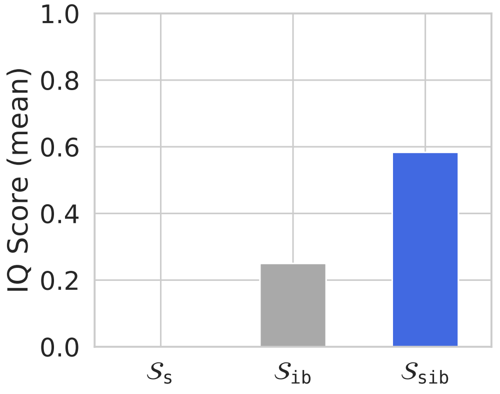


Figure 5.15: GAV mapping quality

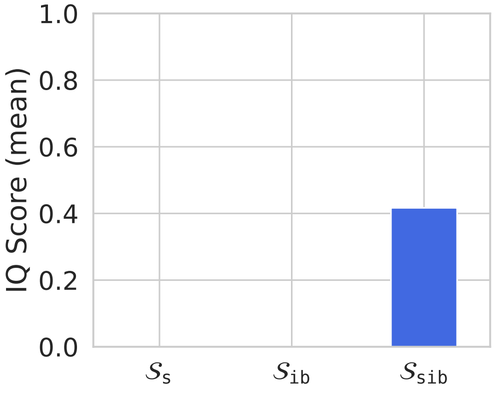


Figure 5.16: GLAV mapping quality

of the IQ-Meter score. Our proposed algorithm also has higher consistency (lower score standard deviation) than the other two algorithms.

In Figures 5.13 through 5.16 we plot the mean quality scores of the three algorithms on the generated scenarios from the four classes: SH-LAV, LAV, GAV, and GLAV. The proposed $\mathcal{S}_{\mathtt{sib}}$ algorithm has the highest quality in all four classes; in particular, in the GLAV class it was the only algorithm to find a mapping with non-zero score.

In Figures 5.17 and 5.18 we plot mean quality scores for $\mathcal{S}_{\mathtt{ib}}$ (grey) and $\mathcal{S}_{\mathtt{sib}}$

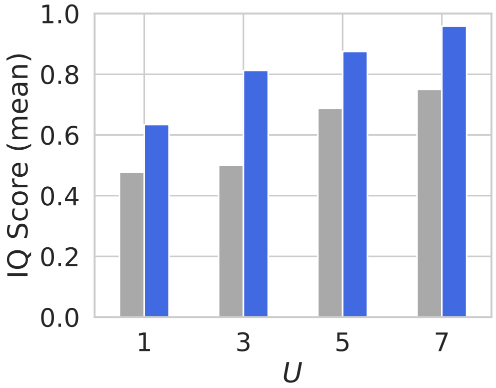


Figure 5.17: Quality while varying U

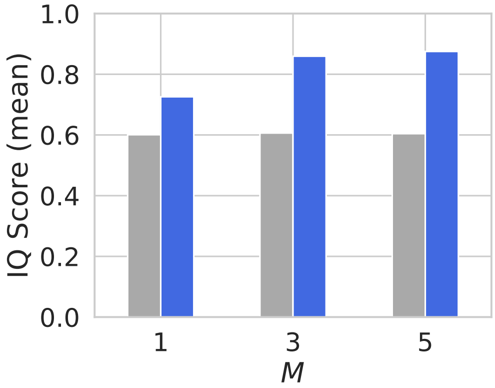


Figure 5.18: Quality while varying M

(dark blue) while varying the algorithm parameters U and M (Section 5.5.3). Recall that U sets the number of iterations of the local search subroutine (Algorithm 2), and that M sets the maximum number of stages of the main loop in $\mathcal{S}_{\texttt{sib}}$ (Algorithm 1) and $\mathcal{S}_{\texttt{ib}}$ (Algorithm 3). The baseline $\mathcal{S}_{\texttt{s}}$ does not use either parameter, so we do not plot its scores. As expected, both algorithms' quality scores improve as we increase the number of iterations in the local search, although our proposed algorithm has higher quality at every setting of U. In contrast, only the score for $\mathcal{S}_{\texttt{sib}}$ increases with M; this is possible because $\mathcal{S}_{\texttt{sib}}$ has the opportunity to repeatedly refine a st tgd over multiple stages, while $\mathcal{S}_{\texttt{ib}}$ starts a new st tgd in each stage.

Our approach ($\mathcal{S}_{\texttt{sib}}$) achieves a perfect quality score of 1.0 on all problems with parameters U and M set as low as seven and three, respectively.

5.7.2 Scalability

In Figure 5.19 we plot the running times of the three algorithms over all generated scenarios and over all scenario parameter settings. As expected, the $\mathcal{S}_{\texttt{s}}$

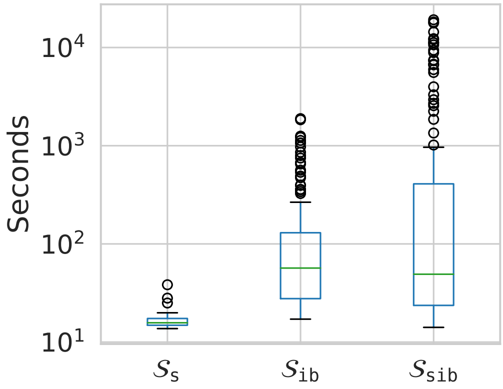
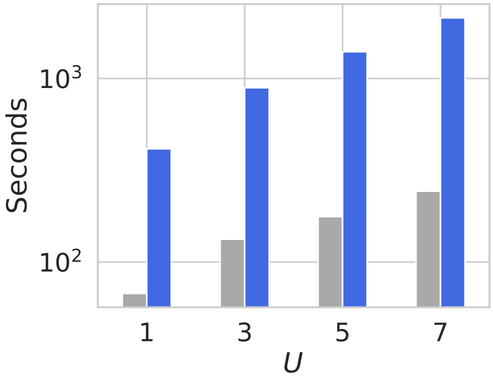


Figure 5.19: Running times



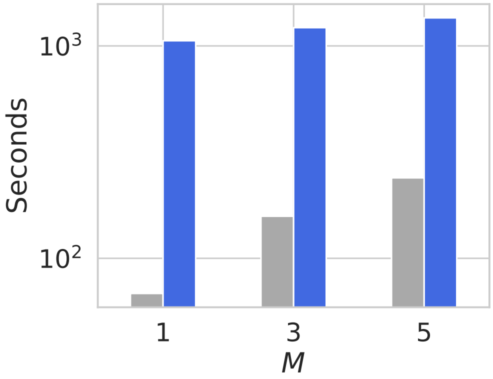


Figure 5.20: Times while varying U

Figure 5.21: Times while varying M

algorithm is the fastest by far, as it does only a single selection step. Algorithms $\mathcal{S}_{\mathtt{ib}}$ and $\mathcal{S}_{\mathtt{sib}}$ take longer and use the same settings for maximum numbers of iterations. On the LAV and GLAV problems, $\mathcal{S}_{\mathtt{sib}}$ takes significantly longer than $\mathcal{S}_{\mathtt{ib}}$; however, on those problems $\mathcal{S}_{\mathtt{ib}}$ fails to find good mappings at all, and stops while $\mathcal{S}_{\mathtt{sib}}$ is still improving its mappings. On Problem 5.14, both $\mathcal{S}_{\mathtt{ib}}$ and $\mathcal{S}_{\mathtt{sib}}$ get the same quality score, but $\mathcal{S}_{\mathtt{sib}}$ is able to finish significantly faster.

In Figures 5.20 and 5.21 we plot the mean running times of algorithms $\mathcal{S}_{\mathtt{ib}}$ (grey) and $\mathcal{S}_{\mathtt{sib}}$ (dark blue) while varying their U and M parameters, respectively.

Both algorithms' running times increase with the parameters, which control the maximum iterations of the search inner and outer loops. Algorithm $\mathcal{S}_{\mathtt{ib}}$ has lower running times because it refines fewer st tgds each stage.

Other than the algorithm choices we've discussed, we did not fully optimize the performance of any of the algorithms. In particular, we did nothing to avoid recalculation of some data preparation for covers, or re-refinement of some st tgd. The data preparation itself could also be made faster with straightforward indexing over the source and target instances. Also, a major advantage of boosting is that the set of homomorphism checks needed to calculate coverage can be reduced by filtering the target instance to the set of tuples that are still not fully explained. Finally, other than checking for zero score when selecting the relation to refine, we did no extra checking for convergence or other early stopping conditions. Any of those additional optimizations could make the algorithms even faster.

5.7.3 Results on Real Data

To complement our evaluation on generated mapping problems, we also test our approach on problems with real data. We map Allen Brain Atlas (ABA), and UniProt (Universal Protein resource) schemas to the Semantic MediaWiki Linked Data Environment (SMW-LDE) Ontology [67, 68]. ABA has one relation and 15 tuples and UniProt has one relation and 15 tuples. The common target schema has 31 relations. We construct data examples from the source instances, and test the ability of search to discover good mappings despite the lack correspondences or foreign keys.

On ABA the search algorithm found an ideal mapping and on UniProt it found a nearly ideal mapping.

The only difference between our mapping and with the gold standard mapping for UniProt was in one argument in one target relation the search placed an existential variable instead of a universal variable. When we inspected the problem inputs, we found that the argument in the data example happened to be entirely filled with nulls. Therefore, the mapping found by the search actually produced the same tuples as the gold standard. When a data example lacks enough evidence to select the gold standard mapping, as in this case, another approach could be to leverage other types of evidence, such as column names. However, in this case the source and target column names (`Mgi_Marker_Accessionid` and `value`) seem unrelated so were unlikely to help any mapping approach produce the gold standard.

5.8 Related Work

Automatic schema mapping discovery is in a broader class of rule learning problems, in which each hypothesis H is a set of rules from language $\mathcal{L}$ and we optimize a score function over those subsets. Other examples of rule learning are inductive logic programming and structure learning [de Raedt et al, 2016]. Most existing approaches either *iteratively* refine a single hypothesis, or work in *two steps*: (1) generating a large set of candidate rules, then (2) selecting a subset. Each has advantages and disadvantages with respect to the several criteria:

Two-step approaches may use different kinds of evidence in each step. In the first step, some approaches use the syntax of hypotheses to generate candidate rules. Some instead take a *bottom-up* approach, using the heuristic that repeating patterns in the data are likely to be elements of good rules [77, 78, 79]. The score function is then used alone in the second step. Iterative approaches leverage syntax to generate refinements [80] and may also leverage data [81]. Iterative approaches then use the score directly to update H.

A hypothesis is ideal if it matches a given gold standard, or is equivalent to the gold standard with respect to the task. Assuming an ideal solution $P \subseteq \mathcal{L}$ exists, finding it with two-step approaches requires $P \subseteq \mathcal{C}$. If P is complex, e.g., having many conjuncts, and bottom-up heuristics do not yield P, generating it using syntax alone may result in more candidates than is feasible in the second step. Finding P iteratively requires sufficient refinements, continuing past local optima, so most approaches use search algorithms, e.g., beam search, or techniques such as random perturbations [82]. By their design, iterative approaches can achieve highly expressive rules more easily than two-step approaches.

The relative running times of two-step and iterative approaches vary by parameter settings. However, the trend in recent two-step systems is that optimization is very efficient, scaling to large numbers of candidates. Additionally, systems with memory needed to optimize over many candidates are increasingly available. Iterative approaches do not generate candidates up front, so use less memory than two-step. However, the work done in each iteration is not as straightforward to accelerate or parallelize, compared to the optimization done in two-step systems.

In summary, two-step approaches leverage a wider range of evidence than iterative approaches, are less likely to get stuck in local optima and are increasingly efficient. On the other hand, iterative approaches more easily discover highly expressive rules. Our proposed approach (Section 5.3) combines benefits of both. Like two-step, we use a set of candidate rules — instead of refining a single hypothesis — then efficiently find the best subset. Like iterative approaches, we use refinements to achieve highly expressive rules. Instead of using the refinements to replace the current hypothesis, we add them to the set of candidates, then repeat the optimization.

5.9 Conclusion and Future Work

In this chapter, we introduce a new approach to guide a search over possible mappings, exploiting a useful property of the objective we introduced in Chapter 3 that allows us to use it to guide both selection over subsets of st tgds and refinement of individual st tgds. Specifically, we use a *boosted search* organized in stages. In each stage, we change the makeup of the mapping – suppressing st tgds, generating new individual st tgds, or both. We generate new st tgds using refinement operators, guided via errors caused by flaws of the current mapping. We also introduce a revised definition for coverage, as well as a novel set of generated scenarios in four categories of mappings. We evaluate the new search approach on generated as well as real data sets, showing its potential to discover high quality mappings despite a variety of metadata flaws. Future work could improve the efficiency of the algorithm further with data indexing, data filtering, and other optimizations.

Chapter 6: Joint Matching and Mapping

6.1 Introduction

Metadata arises from the source and target schemas; another form of metadata are links, called *matches* or *correspondences*, from attributes in our target schema to attributes in the source that share the same meaning, i.e., a *semantic* alignment. For example, if an attribute in the target contains ages of people, correspondences would identify any attributes in the source that also contain ages. Correspondences are helpful for generating possible transformations; for example, widely-used algorithms like Clio [16] use them to derive st tgd mappings.

Unfortunately, correspondences can be noisy, which can result in incorrect mappings. Matching systems can provide confidence levels with each correspondence, but most existing mapping techniques cannot accommodate those confidence levels in a principled way. A significant body of past work focuses solely on the problem of generating correspondences, incorporating a wide variety of useful evidence. Additionally, correspondences are especially helpful when we lack the kind of data example we use in Chapters 3 and 5. An improved ability to handle the noise and uncertainty in correspondences would be very useful, whether used alone or in combination with approaches like those discussed in Chapters 3-5.

One approach is to choose a threshold on the confidence levels associated with correspondences, and use all correspondences with at least that confidence when generating possible st tgds. However, it can be difficult or impossible to find a single threshold that separates correct and incorrect correspondences if confidences are not perfectly accurate or calibrated. Some recent approaches consider a variety of confidence thresholds when generating a mapping, but still rely on the ordering provided by confidence to rank good correspondences above bad ones [24]. This is understandable, because a discrete search over all subsets of correspondences would be intractable.

However, clues to which correspondences are correct may lie in a holistic view of both correspondences and st tgds. I consider the case that some correspondences can be found *jointly* with mappings as a strategy to improve and harmonize both. In this chapter, I propose a joint probabilistic mapping generation model that finds missing correspondences and handles their inherent noise and uncertainty in a principled way.

Like the model in Chapter 3, this approach builds upon probabilistic soft logic (PSL) [6]. PSL uses first order rules to specify probabilistic models in relational domains, and provides an efficient, generic engine for probabilistic inference in these models [83, 53]. Compared to earlier chapters in this dissertation, we assume different inputs; specifically, although we use data instances, we don't assume access to a data example with overlapping instances. This is an important case that complements the approach described Chapters 3 and 5 and is useful for leveraging additional types of metadata evidence. We also use a different mapping objective than the ones in Chapters 3 and 5. Here, we encode a variety of characteristics of good schema mappings in PSL, thus defining a probability distribution over possible mappings that assigns higher probability to those satisfying these characteristics.

To evaluate our proposed approach we first show that our approach is able to find known matches and mappings in real-world data. We then illustrate the flexibility of the approach by evaluating different combinations of rules from our PSL program and show how they contribute to the system's effectiveness. Finally, we show that our system scales well with schema size and complexity.

The rest of this chapter is organized as follows. First, we provide an overview of the problem we are solving and a brief review of PSL. Then we present our approach from start to end. In Section 6.4 we present the results of our experiments using the approach.

6.2 Mapping and Matching Problem

A *schema* is a finite set of relation symbols, corresponding to table names in a relational database. Each relation symbol has an associated arity n and set of n *attributes*, corresponding to column names. An *instance* of a relation of arity n is a finite set of n-tuples, corresponding to a database table. An instance of a schema contains one instance for each of the schema's relation symbols, and corresponds to a database. A relation's *key attributes* are a subset of its attributes whose values uniquely identify each tuple in an instance. We use Datalog notation to represent an instance of an n-ary relation r as a set of ground atoms of the form $r(a_1, \ldots, a_n)$, where the a_i are constants, and a query over r as a non-ground atom $r(X_1, \ldots, X_n)$, where the X_i are logical variables. A *mapping* is a finite set of source-to-target tuple generating dependencies (st tgds) [63], that is, logical rules of the form $\forall \vec{x}, \vec{z}, .\ \phi(\vec{x}, \vec{z}) \rightarrow \exists \vec{y}.\ \psi(\vec{z}, \vec{y})$. Here, ϕ is a conjunction over a *source* schema S, and ψ is a conjunction over a *target* schema T. For convenience, we will omit quantifiers.

Recall from Chapter 3 that an instance of the target is an exchange *solution* for the source schema, a target schema, and a mapping if and only if the source and target instances together satisfy all st tgds in the mapping [17]. In that chapter, we used exchange solutions and the notion of a data example as the basis for the mapping objective and also the final evaluation using IQ-Meter [61]. That is, we used an extensional approach to evaluating mappings, by focusing on their effects on exchange solutions. In this chapter, we instead use an intensional evaluation, in which we directly evaluate the ability of the mapping approach to predict the correct st tgds and matches.

The advantage of this approach – and the reason why it is complementary to Chapters 3 and 5 – is that an intensional approach does not require a data example. For this reason, we will simply use common learning measures, such as false positives and false negatives, to evaluate our approach.

6.3 Our Approach

Our proposed solution is based on PSL, a framework that allows the specification of both hard and soft logical constraints, supports soft truth values, and provides efficient techniques for probabilistic inference. By encoding characteristics of a good schema mapping in PSL, finding a schema mapping becomes a probabilistic inference task (Section 6.3.2). Before the inference step we first generate logical structures called *candidate queries*, taking into account available constraints and data instances (Section 6.3.1). The PSL program then determines how to form st tgds from these queries, jointly inferring matches and st tgds based on both instance data and schema information.

To discover schema mappings with PSL, we construct a program that describes good mappings and then infer a *most probable explanation* (MPE), which is a most probable assignment of truth values to the unknown atoms. One of the reasons PSL is well suited to schema mapping discovery is that MPE inference is a fast, convex optimization [53]. As we show (Section 6.4.3.3), our approach is therefore very scalable. Other features we leverage include PSL's ability to directly integrate similarity functions into the model by treating them as the continuous truth values of logical atoms and to reason with extensive amounts of input as evidence.

Our approach consists of three steps.

1. Generate candidate queries over the source and target schemas (Section 6.3.1).

2. Perform probabilistic inference in the PSL program in Figures 6.1 and 6.2

(Section 6.3.2) to find optimal soft truth values for logical atoms encoding matches and st tgds, i.e., pairs of source and target queries as well as equalities between logical variables in these queries. Higher truth values indicate higher certainty of the corresponding st tgd being part of the final mapping. This step jointly discovers matches (limited to attribute-attribute matches) and st tgds, thus addressing the full mapping task without need to provide matches as input, though those are exploited if available. The second step results in a ranking of st tgds based on their truth values.

3. A user selects the final st tgds from the rankings given in the second step.

In the remainder of this section we describe in detail the steps of our approach.

6.3.1 Generating Candidate Queries

In Chapters 3 and 5, we start with a set of candidate formulas, where each formula is a whole st tgd. In those chapters, our goal is to model the interactions between st tgds and data instances, so st tgds and tuples are the main objects represented in that model.

In this chapter, we focus on correspondences and their special use for connecting the body and head of a st tgd. Therefore bodies, heads, and correspondences are the main objects represented in this model. For that reason, the input formulas we start with here are bodies and heads of st tgds, i.e., candidate conjunctive queries. Those candidate queries are provided as evidence to the PSL program for schema mapping discovery, which selects st tgds that connect source to target queries.

We generate candidate queries for each schema using an approach that a simplified form of the chase algorithm [84] employed in the Clio system for mapping discovery [16]. To generate candidate queries, we start with one query for each source and target relation. We then repeatedly apply foreign key constraints to create conjunctions of atoms. Each conjunction formed at each step becomes a candidate query in the final set. To understand the benefit of this simplified method, compare how the queries are used by Clio and by our approach. Clio uses all pairs of source and target queries, and removes those that (i) do not cover attribute matches provided as input or (ii) are subsumed by some other mapping that covers the same matches. Our approach starts similarly, but pairs the queries using the PSL model. Using PSL to pair the queries gives us the flexibility to generate lots of simple queries, then rely on the PSL model to find an ideal pairing.

6.3.2 Probabilistic Inference in PSL

Our PSL program uses two types of predicates (Section 6.3.2.1). *Closed* predicates, for which the truth values of ground atoms are provided as evidence, represent information on the schemas and their instances, including degrees of similarity between elements of those. Meanwhile, *open* predicates, for which the truth values of ground atoms are determined during probabilistic inference, represent attribute matches and st tgds.

Four types of first-order rules encode dependencies between closed and open predicates, and between different open predicates:

- **Structural evidence rules** propagate characteristics of the structure of schemas and data instances to open atoms that specify matches and st tgds.
- **Similarity evidence rules** propagate degrees of similarity from elements of schemas and data instances to open atoms that specify matches and st tgds.
- **Coverage rules** express the preference for complete mappings, that is, sets of st tgds covering as much of the source and target schemas and data instances as possible. These rules use existential quantification to determine the set of ground atoms they range over based on the evidence. We use prioritized disjunction rules (Chapter 4) to allow the existential quantification, which is otherwise not allowed in the PSL language.
- **Coupling rules** enable joint inference by creating dependencies between atoms of multiple open predicates.

We discuss the rules that are relevant for discovering matches in Section 6.3.2.2, and those that use matches to discover mappings in Section 6.3.2.3.

6.3.2.1 PSL Predicates

Our PSL program uses (1) closed predicates to represent information on the source and target schemas and instances, the queries generated in the first phase of mapping generation, and similarity functions evaluated over the source and target, and (2) open predicates to represent matches and st tgds. To visually distinguish these, we underline the names of open predicates.

All relations in the source and target schemas as well as their attributes are represented by the predicates $\mathrm{sAtt}(R_s, A_s)$ and $\mathrm{tAtt}(R_t, A_t)$, respectively. The key attributes of all target relations are given by $\mathrm{key}(R_t, A_t)$. Our rules do not use key attributes of the source, which are therefore not represented. We further use the predicate $\mathrm{related}(R, A, R', A')$ to provide pairs of attributes (and their relations) from the same source that are related based on structural information, that is, are either in the same relation, or in two relations linked by a foreign key constraint.

For each query generated in the first phase of our approach (Section 6.3.1), the predicates $\mathrm{sQ}(Q_s)$ (for the source) and $\mathrm{tQ}(Q_t)$ (for the target) provide a query identifier. The structure of these queries is represented by the predicate $\mathrm{var}(Q, N, R, A, V)$, whose arguments correspond to the query identifier, the position of a atom within the query, the relation of that atom, an attribute of that relation, and an identifier for the logical variable corresponding to that attribute in the atom. That is, $\mathrm{var}(Q, N, R, A, V)$ denotes that in the Nth conjunct of conjunctive query Q, variable V is in the attribute A of relation R. For instance, a conjunctive source query

$$q_1 = r(X) \wedge s(X, X') \wedge r(X')$$

would be represented as follows:

$$\begin{aligned}\mathrm{sQ}(q_1) \wedge & \mathrm{var}(q_1, 1, r, a_1, x) \wedge \mathrm{var}(q_1, 2, s, b_1, x) \wedge \\ & \mathrm{var}(q_1, 2, s, b_2, x') \wedge \mathrm{var}(q_1, 3, r, a_1, x')\end{aligned}$$

We further link queries to all strings denoting attributes appearing in the query via predicate qHasAttName(Q_s, S).

In addition to these purely schema-based predicates, we use similarity functions over both schema elements (e.g., attribute names) and data instances (e.g., values of an attribute). More specifically, for matching, similarityFunction(S) lists all available such functions, whereas similarBy(S, R_s, A_s, R_t, A_t) evaluates the similarity function given by its first argument on the attributes of relations specified by the remaining arguments. These similarity functions can use schema information, such as attribute and relation names, or compare the actual values taken by the attributes in the data instances. Examples include similar names of attributes, similar sets of relation-attribute name combinations, or similar sets of frequent characters present in data instances. The predicate conditions(S, R_s, A_s, R_t, A_t) allows us to restrict the use of these similarity functions based on the function itself as well as the arguments it is applied to, as we discuss in more detail in Section 6.3.2.2. For mapping, we use a single similarity-based set predicate similarAttNames/2, whose arguments are sets of strings. It is defined as

$$\frac{\sum_{i\in\{Q_s.\text{qHasAttName}\}}\sum_{j\in\{Q_t.\text{qHasAttName}\}} s_p(i,j)}{\max(|\{Q_s.\text{qHasAttName}\}|\,,|\{Q_t.\text{qHasAttName}\}|)}$$

where $s_p(i, j)$ is a function of the similarity of two attribute names and we use PSL's object-oriented syntax for sets, that is, the set $\{Q_s.\text{qHasAttName}\}$ contains all strings S for which qHasAttName(Q_s, S) is true. In our program, the similarity of any two attribute names is available as similarAttName/2, so that predicate is used when defining similarAttNames/2. All closed predicates not based on similarity functions have Boolean truth values.

Our program further uses three open predicates that represent matches and st tgds, whose truth values are to be determined by inference. Matches are represented using match/4, whose arguments are a source relation, an attribute of that source relation, a target relation, and an attribute of that target relation. st tgds are represented using two predicates, tgd/2 and equals/4. An atom of the form tgd(Q_s, Q_t) states that the queries with identifiers Q_s and Q_t (as provided by sQ/1 and tQ/1) form the source and target side of the st tgd, respectively. For any such atom, atoms of the form equals(Q_s, V_s, Q_t, V_t) determine equality between logical variables on the two sides. For instance, given the source query q_1 above and a corresponding representation of a target query $q_2 = t(X'', X''')$, the atoms tgd(q_1, q_2), equals(q_1, x, q_2, x''') and equals(q_1, x', q_2, x'') represent the st tgd $r(X) \wedge s(X, X') \wedge r(X') \rightarrow t(X', X)$.

While we present match as an open predicate here, the implementation of PSL does not make a distinction between open and closed predicates, and thus allows us to provide the grounding of match as evidence (as we do in part of our experiments discussed in Section 6.4), or to even mix the two cases, that is, provide some of the matches as input, and infer the remaining ones. This has a significant advantage for practical applications because we can directly incorporate outputs of external matching systems as observed match atoms.

$$\text{sAtt}(R_s, A_s) \rightarrow \exists(R_t, A_t).\ \text{tAtt}(R_t, A_t) \wedge \underline{\text{match}}(R_s, A_s, R_t, A_t) \tag{a}$$

$$\underline{\text{match}}(R_s, A_s, R_t, A_t) \wedge \text{similarityFunction}(S) \rightarrow \text{similarBy}(S, R_s, A_s, R_t, A_t) \tag{b}$$

$$\begin{aligned}&\text{similarityFunction}(S) \wedge \text{conditions}(S, R_s, A_s, R_t, A_t) \wedge \text{similarBy}(S, R_s, A_s, R_t, A_t)\\ &\quad \rightarrow \underline{\text{match}}(R_s, A_s, R_t, A_t)\end{aligned} \tag{c}$$

$$\begin{aligned}&\underline{\text{match}}(R_s, A_s, R_t, A_t) \wedge \text{related}(R_s, A_s, R'_s, A'_s)\\ &\quad \rightarrow \exists(R'_t, A'_t).\ \text{related}(R_t, A_t, R'_t, A'_t) \wedge \underline{\text{match}}(R'_s, A'_s, R'_t, A'_t)\end{aligned} \tag{d}$$

Figure 6.1: PSL rules for matching

6.3.2.2 Rules for Matching

The first part of our PSL program, Rules (a)–(d) in Figure 6.1, introduces dependencies between the closed predicates providing the evidence and the open predicate $\underline{\text{match}}$ encoding matches.

Rule (a) is a coverage rule, stating that for each attribute A_s of a relation R_s in the source, there should be a match to some attribute A_t of a relation R_t in the target. In this way, it expresses a preference for mappings that include st tgds covering as many source attributes as possible.

Rules (b) and (c) are similarity evidence rules. By using the closed predicates similarityFunction/1 and similarBy/5, we obtain multiple instances of these rules, one for each similarity function provided with the evidence. Inference has to find a trade-off between the constraints induced by all these similarities. More specifically, Rule (b) discourages matches of attributes that are not similar, whereas Rule (c) promotes matches between similar attributes, but only under certain conditions as specified by the closed conditions/5-atoms. For instance, we may encourage matches between attributes with similar names, but only if their data instances overlap.

$$\underline{\text{tgd}}(Q_s, Q_t) \rightarrow \text{similarAttNames}(\{Q_s.\text{qHasAttName}\}, \{Q_t.\text{qHasAttName}\}) \tag{e}$$

$$\underline{\text{equals}}(Q_s, V_s, Q_t, V_t) \rightarrow \underline{\text{tgd}}(Q_s, Q_t) \tag{f}$$

$$\begin{aligned} &\underline{\text{match}}(R_s, A_s, R_t, A_t) \\ &\quad \rightarrow \exists (Q_s, N_s, V_s, Q_t, N_t, V_t).\ \text{sQ}(Q_s) \wedge \text{var}(Q_s, N_s, R_s, A_s, V_s) \wedge \text{tQ}(Q_t) \\ &\quad\ \ \wedge \text{var}(Q_t, N_t, R_t, A_t, V_t) \wedge \underline{\text{equals}}(Q_s, V_s, Q_t, V_t) \end{aligned} \tag{g}$$

$$\begin{aligned} &\underline{\text{tgd}}(Q_s, Q_t) \\ &\quad \rightarrow \exists (R_t, A_t, N_s, R_s, A_s, V_s, N_t, V_t).\ \text{key}(R_t, A_t) \wedge \text{var}(Q_s, N_s, R_s, A_s, V_s) \\ &\quad\ \ \wedge \text{var}(Q_t, N_t, R_t, A_t, V_t) \wedge \underline{\text{equals}}(Q_s, V_s, Q_t, V_t) \end{aligned} \tag{h}$$

Figure 6.2: PSL rules for mapping

Without these extra conditions, the two rules together would establish an equivalence between the truth values of similarBy/5-atoms and corresponding match/4-atoms over the same attributes. That would be an unreasonably strict constraint in practice, as it introduces tension between different similarity functions.

Rule (d) is a structural evidence rule that incorporates schema-based information into the matching process by stating that if a source attribute matches to a target attribute, related source attributes (i.e., in the same relation, or reachable via a join of two relations) likely match to related target attributes.

If all matches are provided as part of the evidence, these rules do not influence probabilistic inference.

6.3.2.3 Rules for Mapping

In addition to the rules for matching, our PSL program contains the rules for mapping given by Rules (e)–(h) in Figure 6.2. These rules introduce dependencies between the evidence, the open predicate <u>match</u> also affected by the matching rules, and the open predicates tgd and equals.

Recall that a tgd atom pairs a query over the source schema with a query over the target schema, and that the associated equals atoms establish which logical variables in these queries are equated, We assume that all equalities between logical variables *within* a query have been established during query generation. As a consequence, each logical variable on the source side should be equal to at most one logical variable on the target side (and vice versa), as equating it with more than one such variable would make the two variables on the target side equal as well. We therefore impose functional constraints on groundings of equals, that is, we only allow a single grounding of equals(Q_s, V_s, Q_t, V_t) with non-zero truth value for any given grounding of Q_s, V_s, Q_t or Q_s, Q_t, V_t.

Rule (e) is an evidence rule influencing the truth value of tgd atoms based on the similarity of the names of attributes appearing in the queries as given by similarAttNames/2. This rule combines two ideas, namely that attributes with similar names often map onto each other, and that many mapping attributes indicate a good st tgd, that is, one that transfers many parts of the schemas. Even though the similarity function does not take into account that attribute mappings should typically be one to one, in practice, it already provides valuable guidance by discouraging st tgds whose queries have dissimilar attribute names. As in the case of the matching rules, we could include additional rules of this type based on other similarity functions. Rule (f) couples truth values of tgd atoms to the truth values of the corresponding equals atoms, expressing a preference for st tgds whose body and head share variables enabling data transfer.

The final two rules are coverage rules, that is, they aim at including as much of the schemas as possible into the mapping. Rule (g) states that if we are confident about a match between attributes, we would like it to contribute to *some* mapping between conjunctive queries. Rule (h) states that for each st tgd, we want it to map *some* source attribute to the key attribute of the target database, given by key(R_t, A_t), so our st tgds will lead to proper tuples in the target.

The weights of all rules are set to 1.

6.4 Evaluation

Our goals are to evaluate whether our approach predicts correct mappings and matches, and to demonstrate the scalability of the system as the size and complexity of the data sources grow. We use STBenchmark to demonstrate scalability because we are able to generate tasks of arbitrary size and complexity.

6.4.1 Measuring the Quality of Mappings

We evaluate the quality of the discovered mapping by comparing its st tgds to the ideal st tgds provided with the evaluation data [41]. Recall that tgd atoms determine which queries participate in the same st tgd, and that equals atoms determine how variables are shared across these queries. We evaluate both aspects as the accuracy of rankings of tgd and equals atoms by the truth values inferred by PSL. Specifically, we calculate the area under the receiver operating characteristic (ROC) curve (AUC) for each. To summarize the accuracy of the approach over an entire benchmark we average the AUC of each task in the benchmark.

For tgd atoms discovered by our program, those that use the same pair of queries (or the same after reordering its query atoms) as a st tgd in the gold standard are considered true positives, and all others false positives. The latter includes st tgds that, even though not in the gold standard, generate correct target data, which would be considered true positives in an evaluation setting based on the quality of target instance data *generated* by a mapping. For this reason we expect that our results would be even more favorable in that evaluation setting.

We separately evaluate the quality of variable bindings of inferred st tgds over correct pairs of queries. We rank the equals atoms associated with true positive tgd atoms. In other words, for each atom equals(Q_s, V_s, Q_t, V_t) in the rankings there must exist a tgd(Q_s, Q_t) in the gold standard with the same queries Q_s and Q_t.

6.4.2 Data Sources

The Neuroscience benchmark [67, 68] provides a real-world test setting that requires integration of four neuroscience data sources with simple schemas into a target Web Ontology Language (OWL) ontology.[1] The STBenchmark system [76] generates synthetic relational schemas and mappings between them based on input parameters controlling the size of the schemas and the complexity of the mappings, and thus allows for controlled experimentation on larger scale schemas.[2] The Amalgam setting [66] considers mappings between manually generated, more

[1] We thank Knoblock et al. [68] for providing their data and mappings.

[2] `http://db.disi.unitn.eu/pages/stbenchmark/`

complex relational schemas for different bibliographic data sources.[3]

6.4.2.1 Neuroscience

The four data sources (Allen Mouse Brain Atlas (ABA), KEGG, PharmGKB, and Uniprot) in the Neuroscience setting contain a total of nine relations. Each relation can be mapped to the target independently of all others, so we consider one mapping discovery task per relation. For each such task, the source schema S contains the corresponding relation only, and the source instance $I(S)$ is the instance of this relation. The common target schema T for all tasks is based on the OWL ontology Wiki.owl. It contains one relation for each of the four classes Disease, Drug, Gene and Pathway, with one key attribute corresponding to an object IRI and additional attributes corresponding to all object and data properties of the class. For each task, the target instance is populated with the output of Knoblock et al.'s mapping of the corresponding relation evaluated on its source instance using Karma[4]. As ABA and KEGG Pathway have been used during development, we evaluate on the tasks for the remaining seven source relations.

The source side queries directly correspond to source relations here, that is, for each task, the evidence contains a single query $r_s(X_1, \ldots, X_n)$ over the relation r_s to be mapped. The target side queries, however, are truly conjunctive. Each OWL class in the target is a relation and has attributes comprising the data and object properties

[3] We thank Miller et al. [66] and Boris Glavic for their version of this benchmark.

[4] `https://github.com/InformationIntegrationGroup/Web-Karma-Public`

6.4.2.2 STBenchmark

In the second setting, we generate mapping discovery tasks using STBenchmark, which we configure to generate source schemas, target schemas and mappings with varying size and complexity. We vary the number of relations and the number of attributes of each relation, which determine the size of the schemas, and the heterogeneity of the schemas, which influences the structural complexity of mappings. For the latter, we consider *copying*, where attributes from a source relation can directly be mapped onto those of a target relation, *vertical partitioning* (VP), where attributes of a source relation are mapped onto attributes of multiple target relations (which requires discovering the correct join on the target side), and *denormalization* (DN), where attributes from multiple source relations are mapped onto attributes of a single target relation (which requires discovering the correct join on the source side). For VP and DN, a join size parameter determines the number of joins on the complex side of the mapping.

Using the benchmark, we generate five tasks. For copying, we consider three relations with two attributes each (*COPY1*) as well as five relations with five attributes each (*COPY2*). For vertical partitioning, *VP1* has five source relations with 10 attributes total, and six target relations with twelve attributes total; *VP2* has one source relation with eight attributes total and four target relations with 14 attributes total. The denormalization task *DN1* has three source relations with a total of seven attributes and one target relation with five attributes.

For both Neuroscience and STBenchmark, our goal is to test the mapping capability and scalability, not matching. Therefore we extract Boolean-valued closed match atoms from the benchmark data manually.

6.4.2.3 Amalgam

The Amalgam benchmark for schema integration consists of independently modeled schemas of similar sources of bibliographic data as well as mappings between those. While the concepts and content in the schemas are similar, their structures are not. We consider mapping from *Schema 1* to *Schema 3* (task *A1*) with 15 relations and 102 attributes in the source schema and five relations and 28 attributes in the target. We also consider mapping from the *DBLP* schema to *Schema 1* where the source schema has seven relations and 46 attributes and the target has 15 relations and 102 attributes (task *A2*).

6.4.3 Results

6.4.3.1 Effectiveness on Benchmark Tasks

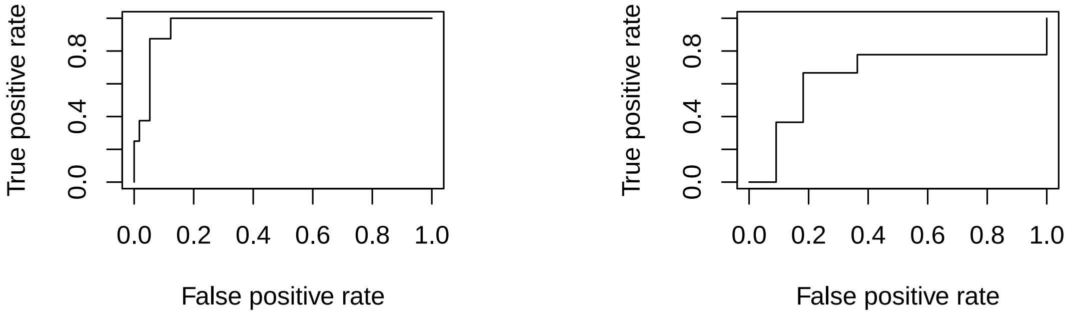
Using our approach, we discover schema mappings on the benchmark tasks. For each task we use the full program in Figures 6.1 and 6.2 as well as the functional constraints.

- On the Neuroscience benchmark, our method discovers perfect st tgds for all seven unseen relations, that is, rankings of both $\underline{\text{tgd}}$ and $\underline{\text{equals}}$ atoms in the MPE state have AUCs of 1.0. Running times vary from less than a second for source relations with few attributes to five seconds at most for larger source relations with eight attributes.

- On STBenchmark, the full program again achieves perfect results on all tasks.
- On Amalgam task *A1*, the the full program achieves good AUCs of 0.93 and 1.0 for tgd and equals atoms, respectively, and running-time is approximately 38 minutes. On task *A2* the full mapping took just over four minutes and achieved AUCs of 0.52 and 0.78 for tgd and equals atoms respectively.

Looking more closely at the two tasks that revealed errors, the false positives on Amalgam task *A1* included st tgds connecting queries over published authors in the source schema to queries over unpublished authors in the target schema. The gold standard does not include these st tgds, perhaps because the authors in the instance data are disjoint. This suggests that combining the PSL program from this chapter with the one in Chapter 3 – which uses the instance data to influence the truth values of tgd atoms – may improve results further.

On Amalgam task *A2*, the errors are due to four st tgds in the gold standard that require a join between relations in the source schema where no foreign key constraints were provided for the join attributes, meaning that our query generation did not propose the required candidate queries. The majority of false positive st tgds discovered by our approach were fragments of these st tgds. Improving the queries and st tgds using the refinement operators and search algorithm from Chapter 5 is a promising approach to increase accuracy further on this task.



(a) ROC for tgd atoms on Amalgam

(b) ROC for equals atoms on Amalgam

Figure 6.3: ROC for tgd atoms (a) and equals atoms (b) found by the mapping program extended with the matching rule using attribute-name similarity on Amalgam task A_1.

6.4.3.2 Effectiveness of Joint Matching and Mapping

For inferring matches and st tgds jointly, we combine the mapping program with the matching rule (c) and two similarity functions, the first based on Levenshtein distances between attribute names, the second comparing sets of frequent n-grams present in the source and target data instances. For both similarity functions, we do not impose further conditions in the rule body. We again use rule weight 1. We inferred match atoms for Amalgam *A1* only; this is because the synthetic schemas generated by STBenchmark are not intended for matching evaluation, e.g., their relation and attribute names are unrealistic. STBenchmark, Neuroscience, and Amalgam *A2* lack at least one of the two target data instances, which is needed for matching rules considering instance data as evidence.

When combining the mapping program with the matching rule (c) and the similarity function based on the attribute names on task *A1*, running time increases to

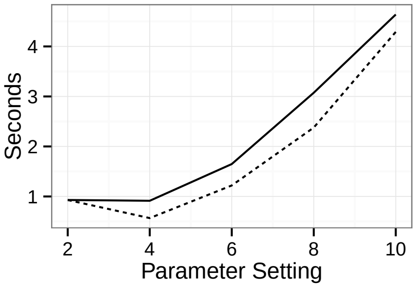


Figure 6.4: Average running time over five random STBenchmark COPY tasks with either two relations with varying number of attributes per relation (dashed line) or two attributes per relation and varying number of relations (solid line).

two and a half hours and the AUCs for tgd and equals are 0.95 and 0.65, respectively. Thus, we retained the same quality of the tgd atoms here as when we used matches extracted by hand. In Figure 6.3a the ROC curve for tgd atoms indicates that the ranking reveals nearly all of the best (gold standard) st tgds quickly; we find approximately 90 percent of the correct st tgds while admitting less than 10 percent of false positives. However, some metadata attributes with identical names made selecting correct equals atoms more difficult (Figure 6.3b).

Despite the simplicity of the matching part of the program, these results demonstrate that jointly inferring matches and st tgds is feasible. Incorporating further matching characteristics is a promising direction for future work.

6.4.3.3 Scalability

We use STBenchmark to generate schema mapping tasks to measure the scalability of our approach. All running times we record are for PSL inference only, and do not include candidate query generation, which typically takes seconds.[5] We

[5] All running times were recorded using a 2.66 GHz Intel Core i7 with 8 GB of RAM.

measure scalability with respect to the size of the schema and the complexity of joins across relations.

Figure 6.4 shows running times for different schema sizes, where we either use two relations with a varying number of attributes each, or a varying number of relations with two attributes each. Values are averaged over five random copying tasks per setting. Running times grow with the total number of attributes.

The third parameter explored in these experiments is the join size, that is, the number of atoms on one side of a st tgd that share variables. For five random repetitions on schemas with five relations with two attributes each, join sizes from two to five resulted in average running-times from two to 70 seconds. However, the more substantial effect is on memory usage; memory usage was well under 0.5 GB for all other tasks, but reached one GB for the largest join sizes in this experiment.

6.5 Conclusion and Future Work

In this chapter we introduced a new schema mapping approach that finds matches jointly with mappings as a strategy to improve and harmonize both. We base the approach on probabilistic soft logic, converting the problem to an MPE inference task in a PSL program that encodes characteristics of good mappings and matches. We demonstrated the promise of our approach with experiments on a range of both real-world and synthetic datasets. We demonstrated the flexibility of our approach and analyzed the contributions of rules by systematically exploring combinations of our rules. In our analysis of results, we found specific cases in which results may be improved through combination with the approaches described in Chapters 3 and 5.

Chapter 7: Statistical Transformations for Source Fusion

7.1 Introduction

In previous chapters I have described approaches to overcome noise and ambiguity to discover the true source data structure to transform to the target schema. However, in some important cases the source structure is inherently lacking, and we need to synthesize missing connections between source relations to provide data fitting the target schema. In these cases, we lack the explicit data and metadata we would need to use a conventional transformation. This situation arises frequently when multiple original sources are ingested into a common repository – sometimes referred to as a *data lake*. When used in combination with other relational data integration methods, such as those we introduce in earlier chapters, a new method to handle problems involving multi-source fusion would have many practical uses.

An important strategy for this problem is to leverage side information accompanying the source tuples representing a level of confidence or probability associated with the tuple. This type of side information is common in some important practical problems. A straightforward approach to fusing sources is then to fuse only high confidence tuples from each source. However, that approach assumes statistical independence among sources, i.e., that a target tuple formed by joining two high probability source tuples is also high probability. In fact, we will see that sources can be dependent. Furthermore, the possibility of dependence may increase with the number of source relations we need to fuse.

Substantial past work has focused on fusing single signals (single attributes) from multiple sources, but very little has focused on the more general relational setting. Other past work in relational data integration assumes that single source relations can be transformed individually, e.g., local-as-view mappings [85]; but that doesn't fit this problem setting.

In this chapter, we will show how – with the addition of side and background knowledge – we can learn a *statistical transformation* for this source fusion problem. Specifically, we will show the advantages of learning a joint probability distribution over source relations – rather than assuming statistical independence – and how to generate target tuples from the joint distribution.

We evaluate our approach on an important practical problem in the cybersecurity domain. Cyberattacks are a growing concern for cybersecurity analysts and administrators of networks in governments, commercial companies, and educational institutions. For example, the average data breach exposes tens of thousands of records, and costs the victim organization millions of dollars to mitigate [86]. Early detection, or prediction, of attack events can reduce these costs, but the average breach still takes nearly 200 days to detect [86]. In this work, we use as input the relational data sources produced by cyberattack *sensors*, and the relational data integration problem is to fuse the sensor-based sources and produce consolidated relational detection outputs.

7.2 Problem and Background

There has been progress to develop automated sensors for detecting cyberattack events [87, 88, 89]. To improve the utility of the sensors, *sensor fusion* has the goal of linking and fusing the output data sources generated by the sensors. In this chapter, we use the terms *source fusion* and sensor fusion interchangeably because the origin of the sources are sensors. *Structured prediction* has the goal of producing a set of accurate and detailed event predictions, combining the strengths of the sensors (Section 7.2). We use the terms *event* and *tuple* interchangeably because tuples represent events in our evaluation. Typical transformation rules used in relational data integration, such as st tgds, are not flexible enough to handle confidence side information associated with source tuples. We propose a new approach to *solve the sensor fusion and structured prediction problems jointly.* We do this by learning statistical transformation rules trained over a data example, and exploiting three insights about real cyberattack events' timing and details – which we refer to as *roles*:[1] (a) roles of events are interdependent, (b) events occur in clusters, and (c) events evolve over time (Section 7.3).

Based on these three insights about the structure of attack events, we refer to a timeline of attacks as a cyberattack event network (CEN). We introduce the *event-relational* model using statistical dependencies capturing the three insights, enabling fusion and prediction via collective reasoning over all sensors and events (Section 7.4).

[1] Attributes of events are commonly referred to as *roles*

Category	Output Type	Source	Uses
Network	IP address	Recent network-based attacks	Block communications with IP address
Network	Network port or protocol	Recent attacks	Monitor activity using port or protocol
Host	Hash or binary signature	Detected malicious files	Scan systems for hash code or signature
Host	Host name	Computer targeted in attack	Monitor host for further attacks
Email	Words in email subject	Recent malicious emails	Monitor emails for similar words
Email	External address domain	Senders of malicious emails	Block emails from sender domain

Table 7.1: Example sensor output types, possible sources, and typical uses by cybersecurity analysts or network administrators.

We apply the general event-relational model to CENs; we refer to our implementation using probabilistic soft logic [6] as Cyber Event Relational Fusion (CERF) (Section 7.5). We conduct an extensive empirical evaluation of CERF using a database with nine months of real cyberattacks against a large corporation in the United States (US). We show how CERF fuses sensors to increase the level of detail in predictions. We show that CERF increases accuracy of predicted events by three percent, as measured by area under the receiver operator characteristic (AuROC) curve, and more than doubles maximum lift for high-confidence predictions. We show that, with as little as 10% partially observed events, we increase AuROC by an additional four percent (Section 7.6).

In the remainder of this section, we provide background on current cyberattack sensors, the sensor fusion problem, and techniques for structured prediction.

7.2.1 Cyberattack Sensors and Sensor Graphs

Cybersecurity analysts and network administrators have long used a variety of cyberattack sensors to detect, prevent or mitigate attacks. In Table 7.1, we list example sensor outputs, sources used by sensor software to produce those outputs, and typical uses by analysts. There has been recent progress to use a wider variety of

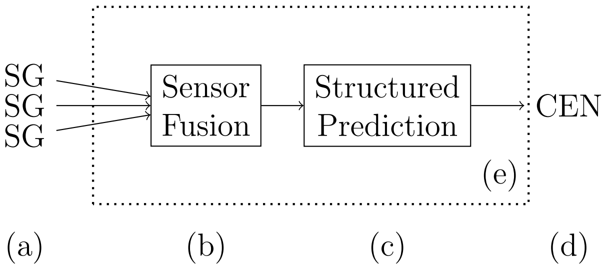


Figure 7.1: Sensor fusion combines sensor graphs (SG). Structured prediction produces a cyberattack event network (CEN). Our proposed approach (dotted) combines both.

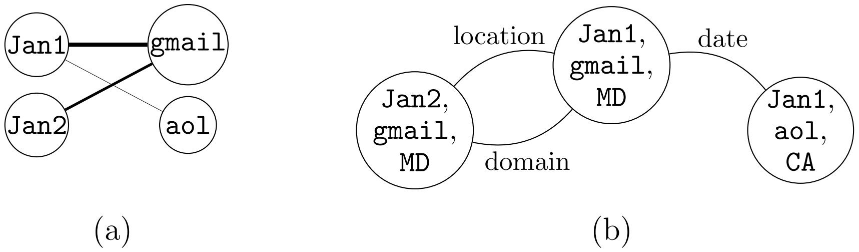


Figure 7.2: Examples of (a) a SG for email domain role; and (b) a CEN with three events, $K = 2$ roles, and three similarities.

input sources, and produce a wider variety of output types [87, 88, 89]. For this work, we assume that sensors produce an output collection every time step, and provide *confidence* levels with each output. Many sensor outputs are indicators that analysts then need to interpret; we instead explicitly *predict* events. Ideally, predictions are both *accurate* and *differentiated*. Accurate predictions reveal true attacks while minimizing false positives. Differentiated predictions identify multiple characteristics of an attack.

Example 8: Predictions of phishing emails, a common type of attack, are *differentiated* if they combine predicted time with details such as the sender address domain, a word appearing in the subject line, and the work location of the recipient. Using differentiated phishing predictions, network administrators can make precise email filters, and search efficiently for related attacks.

While some sensor outputs are simple scalar time series, we focus on structured outputs containing details that can be combined into differentiated predictions. We refer to these dynamically-changing structures as *sensor graphs* (SG) (Figure 7.1a). Even SGs typically contain only a single type of output, e.g., only IP addresses or only hash codes, because each sensor uses just one or a few types of input data. Therefore we assume a SG is a bipartite graph comprising nodes for each time step and *label* (e.g., a specific IP address) for some output type. Edges between time steps and labels represent sensor outputs, weighted by confidence level (Figure 7.2a).

7.2.2 Sensor Fusion

Sensor fusion (Figure 7.1b) is the task of linking and combining sensor graphs. A challenge is that SGs from separate sensors may have no nodes in common with the exception of time steps. Linking shared dates across SGs is a starting point, but does not reveal how to link label nodes across SGs to form differentiated predictions. Learning cross-sensor dependencies is one way to link labels (Section 7.4.2).

7.2.3 Structured Prediction

We represent events and their details as *structured objects*; each is a K-ary tuple $(r_1, \ldots, r_K)$. We refer to details of an event as its *roles*, as they identify participants and objects involved. The value for each role is some constant from a predefined set of labels.

Structured prediction is the task of predicting structured objects; in our case, we predict cyberattack events from the results of sensor fusion (Figure 7.1c). Typically, we make sets of predictions covering some time period. Furthermore, we use the roles of events to form a network of related events, which we call *cyberattack event networks* (CEN) (Figures 7.1d and 7.2b). Techniques for structured prediction are well studied [45, 90, 91], and applied to a number of tasks, which we review briefly here.

7.2.3.1 Predict a Single Role

Predicting even a single role of an event requires sophistication. For example, recent multi-label classification approaches aim to leverage relationships between labels when assigning them to a role. Relationships between labels in a single role include pairwise similarity, organization into categories, or logical constraints [92]. For cyberattack events, we use similar relationships between labels (see Sections 7.5.3.2 and 7.5.3.3).

7.2.3.2 Predict Multiple Roles

Other tasks involve multiple roles. For example, the goal of event sequence label learning is to predict the composition of labels over a collection of roles. Riedel et al. do this with statistical relational learning [93]; we use a similar technique to predict composition of cyberattack roles (see Section 7.4.2).

7.2.3.3 Predict Multiple Events

Predicting multiple structured objects at once – each with multiple roles – can have advantages such as improved accuracy. For example, collective inference [60, 59, 94] and hierarchical tensor representations [95] have been used to generate accurate recommendations with two or more roles. Like Kouki et al., we use collective inference to predict clusters of events within time steps (see Section 7.4.3) and over time (see Section 7.4.4).

Although sensor fusion and structured prediction have distinct goals, we will solve them *jointly* (Figure 7.1e), leveraging the graphical structures of SGs and CENs. We describe CENs in detail in Section 7.3.

7.3 Cyberattack Event Networks

In this section, we describe CENs, our motivating problem setting. We will illustrate three key insights about real attacks using a dataset of attacks events, which will be described fully in Section 7.6.1.1. We refer to this dataset as the ground truth (GT) CEN.

7.3.1 Roles of Events are Interdependent

We see in real cyberattack events that their roles are interdependent. For example, in Example 8 the email subject may have been crafted to target victims at that location to appear more legitimate. To confirm this intuition empirically in the GT CEN, we calculate the Kullback-Leibler divergence $D_{\mathrm{KL}}(p||p')$ where $p(r_1, r_2, \ldots, r_K)$

is the empirical joint distribution of roles in the CEN, and p' assumes independence: $p'(r_1, r_2, \ldots, r_K) = p(r_1)p(r_2)\cdots p(r_K)$. This measure (total correlation [96]) for the GT CEN is 2.1, which is over 70% higher than if role values were shuffled across events in the training set, averaged over five trials. Although unsurprising, this confirms the need for dependencies between roles when linking SGs to form predicted events. In Section 7.4.2 we define a model with these dependencies (event-propositional).

7.3.2 Events Occur in Clusters

We also find that clusters of similar events tend to occur together, even in the same time step, e.g., on the same day. We confirm this using a sample of the GT CEN, which has over one thousand events. To produce the sample, we define an initial set of edges between events representing similarities such as events having the same location (see Figure 7.2b). To focus on individual time steps, we remove edges relating events across multiple days. We combine all parallel edges, resulting in edge weights between one and four. A weight of one means two events occur on the same day; additional similarity results in higher weights. As shown in Figure 7.4, most events on the same day are similar with respect to one or more measures. In Figure 7.3, we plot a five-name snowball sample [97, 98] on 500 seed events with a force-directed layout. Removing cross-day edges causes separate connected components for different days.

We arbitrarily chose one role – location – to color the nodes, which reveals that locations are distributed non-uniformly over days. For example, the component

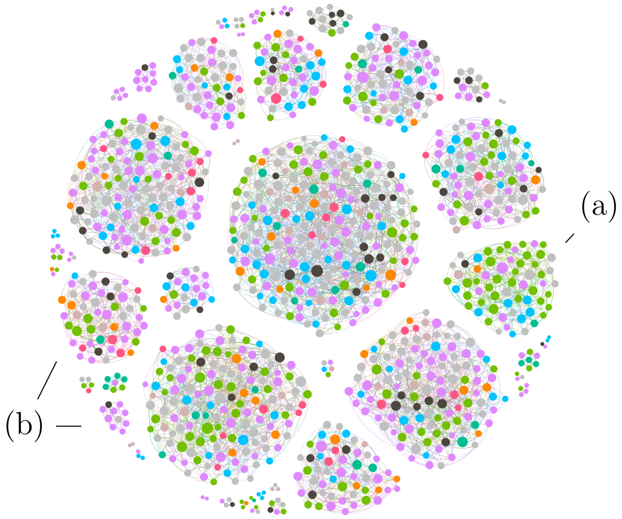


Figure 7.3: Sample of a real CEN, colored by victim location.

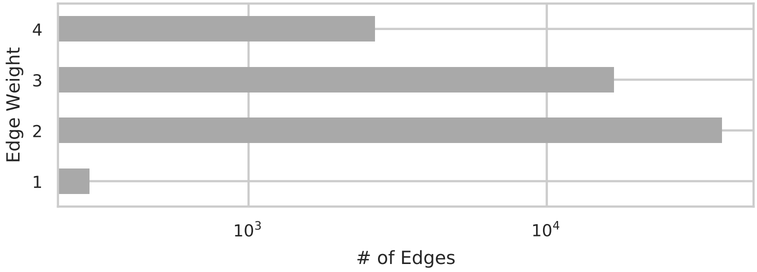


Figure 7.4: Edge weight distribution for graph in Section 7.3.2.

labeled (a) in Figure 7.3 has a high proportion of attacks in California (green), while on other days (b) attacks in California are rare. This pattern, known as homophily [99], is frequently seen in social networks. In Section 7.4.3, we define a model with this dependency (time-propositional).

7.3.3 Events Evolve Over Time

Cybersecurity analysts have observed that attack events tend to occur in clusters over time, progressing through stages. For example, according to the Cyber Kill Chain® framework, exploitation attacks follow delivery attacks, which follow

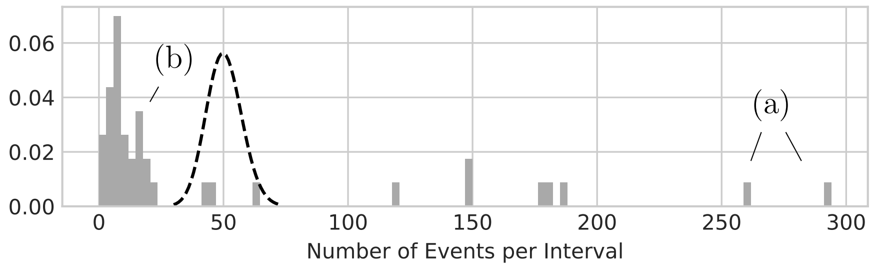


Figure 7.5: Distribution of actual events per interval (grey), and assuming independent arrivals (dashed).

reconnaissance attacks [100]. We consider a related but simpler form of evolution over time: whether events tend to occur in clusters of consecutive time steps. We confirm this in the GT CEN by counting events occurring in fixed intervals; we arbitrarily chose seven days. We compare that to a Poisson distribution resulting from assuming independent arrival times, i.e., not clustered. We plot both in Figure 7.5, which shows that events tend to cluster in large numbers in some intervals (e.g., Figure 7.5a), leaving few in other intervals (b). That is, if we have high confidence in some attack occurring in a time step, it is likely that other attacks occur in surrounding time steps. In Section 7.4.4, we define a model with this dependency (event-relational).

7.4 Event-Relational Model

In this section, we introduce the event-relational structured prediction model, which will be a framework to encode the insights from Section 7.3. For ease of understanding, we lead up to the event-relational model in four steps; in each step we define a model that extends the previous. The most basic of the four is *role-propositional*, which uses a single sensor as input and predicts a single role of a single

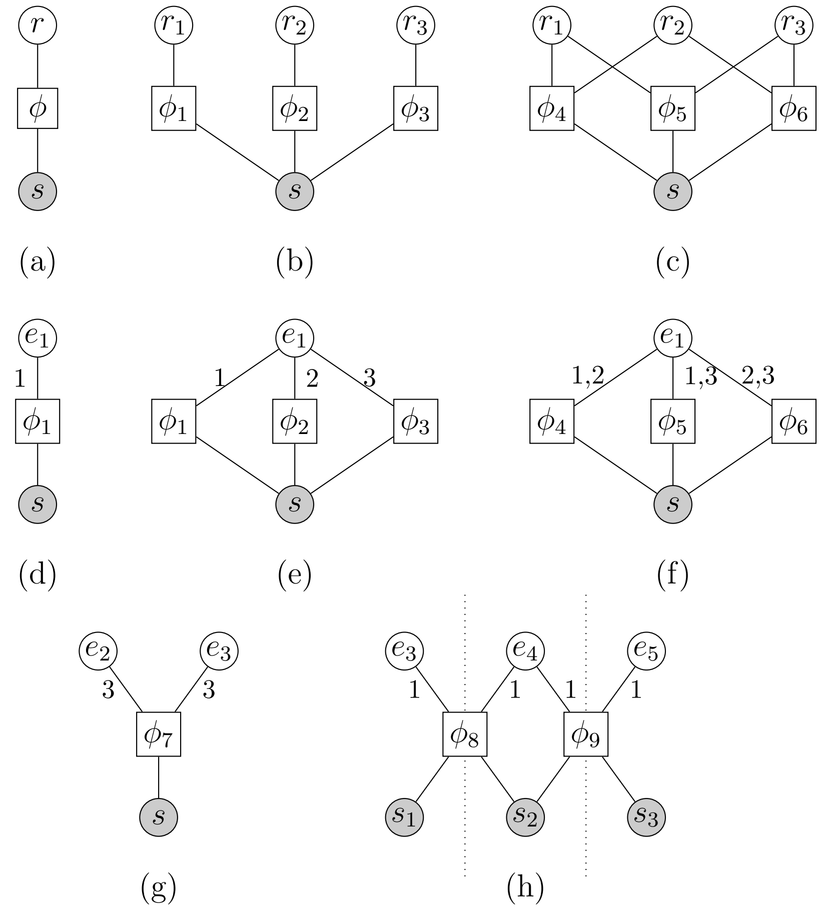


Figure 7.6: Examples of four models: Role-propositional (a). Event-propositional (b and c). Time-propositional rewrites of previous (d-f). Time-propositional with two events (g). Event-relational (h) with three time steps (dotted).

event. The remaining three models extend the first, and each captures one of the three insights:

- Roles of events are interdependent ⇒ *Event-Propositional*
- Events occur in clusters ⇒ *Time-Propositional*
- Events evolve over time ⇒ *Event-Relational*

We define the models in the following four sections.

7.4.1 Baseline: Role-Propositional

A sensor produces an output vector $s \in \mathcal{S}$ that corresponds to some *role* of an event represented by the random variable (RV) r, where the value for r is a label from set $\mathcal{L}$. Predicting r is a multiclass classification problem. We capture the statistical dependency between the sensor output and r using feature function $\phi : \mathcal{L} \times \mathcal{S} \rightarrow \mathbb{R}_{\geq 0}$, a technique commonly used in probabilistic models [45, 90]. Our definition for ϕ is problem-specific; generally, it measures *incompatibility* between a predicted value of r and what we observe in sensor data. Given $\mathcal{L}$ and s, we pick an optimal value for r as $\arg\min_{r \in \mathcal{L}} \phi\left(r, s\right)$. We refer to this model as *role-propositional*, and it represents a baseline approach. We illustrate the model in Figure 7.6a, with shading indicating the observed variable.

Example 9: Suppose role r represents the port targeted in a network-based attack event, and the set of possible labels contains commonly used ports, e.g., $\mathcal{L} = \{\mathsf{22}, \mathsf{53}, \mathsf{80}\}$. Given a sensor output reporting a high confidence of .9 for an attack on port 22 at time step t, feature functions used in our approach would produce a high incompatibility score for any assignment to r other than 22, which has zero incompatibility. The optimum prediction is $r = \mathsf{22}$.

7.4.2 Extension: Event-Propositional

A limitation of the role-propositional model is it uses a single input sensor and predicts events with a signal role, which are not well-differentiated. We will change

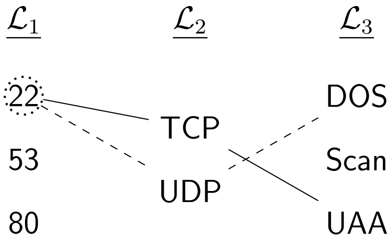


Figure 7.7: Selected role values from sets $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$ in predictions from examples 9 (circle), 10 (dashed), and 11 (solid).

the model as follows:

- Extend s and $\mathcal{S}$ with partitions for additional sensor outputs
- Vector $\mathbf{r} = (r_1, \ldots, r_K)$ replaces the single role r
- K sets $(\mathcal{L}_1, \ldots, \mathcal{L}_K)$ – one for each role – replace $\mathcal{L}$
- Set $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_p\}$ of feature functions replaces ϕ

This *event-propositional* model fuses K sensors and predicts K roles. In Figure 7.6b, we illustrate an example with $K = 3$.

Example 10: We extend Example 9, adding sensors for network protocol and attack class producing high-confidence outputs `UDP` (User Datagram Protocol) and `DOS` (denial of service), respectively. The optimum with respect to ϕ_1, ϕ_2, and ϕ_3 is $\mathbf{r} = (r_1, r_2, r_3) = (22, \texttt{UDP}, \texttt{DOS})$. In Figure 7.7, we illustrate the three sets $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$, and the selected roles from our prediction (dashed line).

Feature functions are flexible in the event-propositional model, and can capture relationships *between* roles. For example, in Figure 7.6c, we use function $\phi_4 : \mathcal{L}_1 \times \mathcal{L}_2 \times \mathcal{S} \to \mathbb{R}_{\geq 0}$. This allows us to predict the most likely *composition* of an event, which may differ from the optimal assignment according to individual sensors.

Example 11: Although assignment $\mathbf{r}$ in Example 10 is optimal with respect to functions ϕ_1, ϕ_2, and ϕ_3, that combination is improbable in this domain. Assignment $\mathbf{r} = (22, \texttt{TCP}, \texttt{UAA})$ is more probable because the Secure Shell (SSH) service running on port 22 uses TCP (Transmission Control Protocol), and – as a remote access service – unauthorized access attempts (UAA) against SSH are more likely than DOS attacks. In Figure 7.7, the solid line marks our updated prediction with the more compatible assignment.

As shown in the example, the best assignment may require balancing functions encoding sensor outputs (ϕ_1, ϕ_2, ϕ_3) and functions encoding compatible event composition (ϕ_4, ϕ_5, ϕ_6). The event-propositional model optimizes all functions in $\mathbf{\Phi}$ simultaneously, forcing assignments to consider all sensors and all types of compatibility *collectively.* See Section 7.5.4.2 for an extended example.

The feature functions in $\mathbf{\Phi}$ may need to be weighted differently, e.g., to tune the balance between sensors and compatibility. We train a weight vector $\mathbf{w} \in \mathbb{R}^p_{>0}$ (see Section 7.5.2.1), associate one weight from the vector $\mathbf{w}$ with each function, and find optimal assignment $\mathbf{r} \in \mathcal{L}_1 \times \ldots \times \mathcal{L}_K$ as $\arg\min_{\mathbf{r}} \mathbf{w}^\intercal \mathbf{\Phi}(\mathbf{r}, s)$.

7.4.3 Extension: Time-Propositional

Although the event-propositional model gives the most likely composition of an event, represented by $\mathbf{r}$, it is limited to a single event. It also doesn't indicate *whether* an event will occur. We will replace the single event $\mathbf{r}$ with a more general binary encoding. In the encoding, an element of vector $\mathbf{e} = (e_1, \ldots, e_u)$ exists for each of u

possible configurations of roles in $\mathcal{L}_1 \times \ldots \times \mathcal{L}_K$ and e_i is true iff its corresponding configuration of roles occurs in an event. Because this models all events in a time step, we refer to this model as time-propositional. We rewrite the event-propositional feature functions for the binary representation; they become $\mathbf{\Phi} : \mathcal{S} \times \{0,1\}^u \to \mathbb{R}_{\geq 0}$ and defined over projections of events. We illustrate rewriting in Figure 7.6d-f, in which an edge annotated with a set $\gamma \subseteq \{1, \ldots, K\}$ refers to a feature function defined over an event e, but using only a projection of e's roles, i.e., $\pi_\gamma(e)$ using relational algebra.

Example 12: Logical atoms are a convenient representation for $\mathbf{e}$, so we use the K-ary logical predicate *event* to represent all events, where the number K of logical terms representing roles varies by the type of attack. CERF also uses this representation (Section 7.5.2). Extending Example 11, we rewrite assignment $(r_1, r_2, r_3) = ($22, TCP, UAA$)$ as $e_1 =$ event(22, TCP, UAA). Rule ϕ_4 in Figure 7.6c places dependencies on r_1 and r_2; the equivalent function in Figure 7.6f uses projection $\pi_{1,2}(e_1)$ to do the same with the 1st two roles of the event, e.g., assigning low incompatibility between 22 and TCP.

Handling multiple events is the goal of time-propositional models; we illustrate a multi-event example in Figure 7.6g.

Example 13: As shown in Section 7.3.2, similar events tend to cluster. Suppose we predict a phishing event $e_2 =$ event(gmail, timesheet, MD) (see Example 8). Also, suppose we have similarities across location labels, e.g., between Maryland (MD) and Virginia (VA) (see Section 7.5.3.2). Function ϕ_7 could add a dependency linking our confidence in e_2 with confidence in an additional similar event $e_3 = \text{event}(\texttt{gmail}, \texttt{timesheet}, \texttt{VA})$.

Given s and $\mathbf{w}$, we find optimal $\mathbf{e} \in \{0, 1\}^u$ as $\arg\min_{\mathbf{e}} \mathbf{w}^\intercal \mathbf{\Phi}(\mathbf{e}, s)$.

7.4.4 Extension: Event-Relational

To capture dependencies across time steps $t_1, \ldots, t_n$, we replace s with vectors $\mathbf{s} = (s_1, \ldots, s_n)$. We refer to this model as event-relational and illustrate an example for $n = 3$ in Figure 7.6h.

Example 14: Clusters of attacks tend to continue over multiple time steps (Section 7.3.3). We extend Example 12 and assume that in step t_1 event e_3 occurs, which we now represent as $\text{event}(t_1, \texttt{gmail}, \texttt{timesheet}, \texttt{VA})$. Function ϕ_8 adds a dependency between e_3 and e_4, where e_4 is identical except it occurs in t_2.

In summary, the following four parameters (counts) characterize the size and complexity of an event-relational model:

- $\underline{K}$ roles in each event
- $\underline{u} = |\mathbf{e}| = |\mathcal{L}_1 \times \ldots \times \mathcal{L}_K|$ possible events in each time step
- $\underline{n}$ time steps
- $\underline{p} = |\mathbf{\Phi}|$ feature functions

With the event-relational model, we find an optimal $\mathbf{e}$ as follows:

$$\arg\min_{\mathbf{e} \in \{0,1\}^{nu}} \mathbf{w}^\intercal \mathbf{\Phi}(\mathbf{e}, \mathbf{s}) \tag{7.1}$$

Although solving Equation (7.1) exactly is NP-hard, in Section 7.5 we describe how CERF finds an approximate solution efficiently. We have shown how the event-relational model enables statistical dependencies including all types shown in Figure 7.6d-h, which capture each of the insights described in Section 7.3.

7.5 Cyber Event Relational Fusion

In this section, we describe CERF, our system applying the event-relational model from Section 7.4 to CENs (Section 7.3).

7.5.1 Probabilistic Soft Logic

We use probabilistic soft logic (PSL) [6] to implement the event-relational model. Using PSL has several advantages: (a) PSL uses logic, a natural representation for roles, events and feature functions. (b) Solving Equation (7.1) exactly is NP-hard, but PSL provides a high quality approximation. (c) PSL scales well to large CENs.

7.5.2 Mapping to Event-Relational Model

We use logical atoms to represent event set **e** (see Section 7.4.3). We use the following mapping to produce the model in PSL:

- Roles ⇒ Logical terms (logical variables or constants)
- Events ⇒ Logical atoms
- Sensor output ⇒ Ground logical atoms

7.5.2.1 Inference and Learning

A PSL program defines a hinge-loss Markov random field (HL-MRF), where weighted logical rules are feature functions $\mathbf{\Phi}$ and atoms with soft truth values are RVs. Maximum a posteriori (MAP) inference in the HL-MRF allows a highly efficient approximation [6]. This is done in PSL by approximating the following, where $\mathbf{x}$ and $\mathbf{y}$ represent observed and inferred atoms, respectively:

$$\arg\min_{\mathbf{y}\in[0,1]^v} \mathbf{w}^\intercal \mathbf{\Phi}\left(\mathbf{y}, \mathbf{x}\right)$$

A high quality binary solution $\mathbf{y} \in \{0,1\}^v$ is possible by applying conditional probabilities rounding to the soft-valued output [6]. In CERF, $\mathbf{x} \equiv \mathbf{s}$, $\mathbf{y} \equiv \mathbf{e}$, and $v \equiv nu$, establishing equivalence with our objective in Equation (7.1). We learn $\mathbf{w}$ with maximum likelihood estimation [53].

7.5.2.2 Soft Truth Values

In addition to the binary solution, the soft-valued $\mathbf{e} \in [0,1]^{nu}$ from PSL has advantages: We can present high-value events to analysts, and measure AuROC and lift (Section 7.6).

7.5.3 Predicates

Logical predicates define the inputs, outputs, and latent variables of a PSL program. We use the following three predicates:

- *event*: a CEN node; its first term is time step T; the remaining K terms vary by event type
- $\sigma_i(R, R')$: a CEN edge – similarity of $R \in \mathcal{L}_i$ and $R' \in \mathcal{L}_i$
- $\mathrm{s}_i(T, R)$: an edge of a SG for role i

MAP inference assigns truth values of inferred event atoms. Truth values of all observed event atoms (see Section 7.6.4) are set to 1.0.

7.5.3.1 Sensors

Truth values of s_i atoms are based on the confidence of the sensor for those outputs (Section 7.6.1.3).

7.5.3.2 Similarities

All similarity σ_i atoms are observed; their truth values are set by external similarity functions.

7.5.3.3 Categories

We use sets of categories over labels within roles. Binary predicate $\succ$ represents membership in a category; $R_i \succ \ell$ is true iff label $R_i \in \mathcal{L}_i$ is a member of category $\ell \in \chi_i$, where set χ_i contains all categories for role i.

Categories have several benefits: (a) They add sensor information to labels. (b) They control the number of parameters (weights) and over-fitting for template-based rules (see Section 7.5.4.1). (c) They help CERF scale to larger CENs with higher values for K by controlling the number of groundings of template-based rules.

7.5.4 CERF PSL Rules

PSL rules implement feature functions of the event-relational model. The simplest rule is the prior $\neg\text{event}(T, R_1, \dots, R_K)$, i.e., any attack is individually improbable. The remaining CERF rules are as follows, where rules with parameterized weights, e.g., $w_{\text{TP}}(i)$, are *templates*:

$$
\begin{aligned}
w_{\text{RP}} :\ & \bigwedge_{i \in 1,\dots,K} \mathrm{s}_i(T, R_i) \rightarrow \text{event}(T, R_1, \dots, R_K) && (\mathsf{RP}) \\
w_{\text{EP}}(i, j, \ell, \ell') :\ & \neg\Big(\text{event}(T, R_1, \dots, R_i, \dots, R_j, \dots, R_K) \\
& \quad \wedge (R_i \succ \ell) \wedge (R_j \succ \ell')\Big) && (\mathsf{EP}) \\
w_{\text{TP}}(i) :\ & \text{event}(T, R_1, \dots, R_i, \dots, R_K) \wedge \sigma_i(R_i, R_i') \\
& \wedge R_i \neq R_i' \rightarrow \text{event}(T, R_1, \dots, R_i', \dots, R_K) && (\mathsf{TP}) \\
w_{\text{ER}} :\ & \text{event}(T, R_1, \dots, R_K) \wedge \sigma_t(T, T') \\
& \wedge T \neq T' \rightarrow \text{event}(T', R_1, \dots, R_K) && (\mathsf{ER})
\end{aligned}
$$

Rule (RP) is role-propositional and assumes roles are independent, combining confidence levels from sensors for each role with a simple conjunction. Rule (EP) is event-propositional; it penalizes incompatible combinations of categories (Section 7.5.3.3). Rule (TP) is time-propositional, propagating confidence across similar events in the

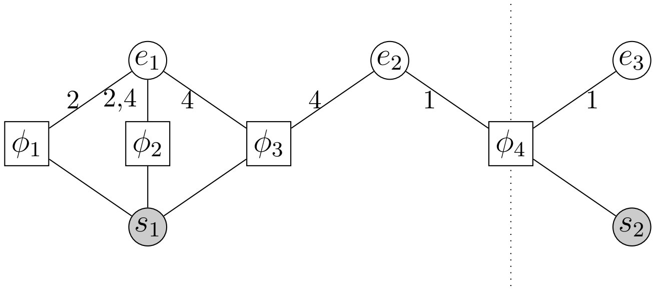


Figure 7.8: Fragment of a CERF model with $n = 2$ and examples of Rule (RP) (ϕ_1), Rule (EP) (ϕ_2), Rule (TP) (ϕ_3), and Rule (ER) (ϕ_4).

same time step. Rule (ER) is event-relational; it is a variant of Rule (TP) that crosses time steps.

7.5.4.1 Partial-Grounding

Rules TP and EP are templates; partial-grounding is the process of setting constant values for each template parameter, e.g., i in $w_{\text{TP}}(i)$. We do partial-grounding as follows: Create a copy of Rule (TP) for every role $i \in 1, \ldots, K$ to learn separate weights for each role. Using category sets $\chi_1, \ldots, \chi_K$ (Section 7.5.3.3), create a copy of Rule (EP) for every 4-tuple in the following set:

$$\left\{(i, j, \ell, \ell') \text{ with } (i, j) \in [1, \ldots, K]^2,\ i \neq j,\ (\ell, \ell') \in \chi_i \times \chi_j\right\}$$

7.5.4.2 Collective Inference

To illustrate how rules combine for collective inference, in Figure 7.8 we present a small, notional fragment of a CERF model. RP rule ϕ_1 allows sensor output in s_1 for the 2nd role (i.e., r_2) to influence the 2nd role of e_1. EP rule ϕ_2 also influences role 2 by preferring compatibility between it and the 4th role. TP rule ϕ_3 is satisfied if e_1 clusters with another similar event e_2 that varies with respect to role 4. ER rule ϕ_4 is satisfied if event e_2 continues in time step 2. Sensor outputs used in any rule can have a global effect on the predicted CEN via this chain of dependencies. For example, if event e_3 is unlikely according to sensor data, that can cause CERF to change its predicted roles for event e_1.

7.6 Evaluation

We evaluate CERF on a real corporate database of cyberattacks. Our goals are to evaluate the following:

- *Accuracy*: measured using AuROC and lift (Section 7.6.3)
- *Partially-observed events*: their effect on accuracy (Section 7.6.4)
- *Rules*: their relative contributions to accuracy (Section 7.6.5)
- *Scalability*: running time of CERF (Section 7.6.6)

Predicting well-differentiated events is another goal. Sensors produce one role each, so CERF's output is K times more detailed. This factor of improvement impacts the cost of solving Equation (7.1), but we show CERF scales well to $K = 4$ (Section 7.6.6).

#	Role (Variable)	Label Set
1	Class of attack (C)	$\mathcal{L}_1 = \{\texttt{Phish}, \texttt{Malware}\}$
2	Job category (J)	$\mathcal{L}_2 = \{\texttt{Acquisitions}, \ldots, \texttt{Training}\}$
3	Location (L)	$\mathcal{L}_3 = \{\texttt{AK}, \texttt{AL}, \ldots, \texttt{WI}\}$
4	Grade/level (G)	$\mathcal{L}_4 = \{\texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{C}\}$

Table 7.2: Event roles in the GT CEN

7.6.1 Data

We describe the GT CEN collected from internal organization records (Section 7.6.1.1), and the input SGs collected from a variety of external data sources (Section 7.6.1.3).

7.6.1.1 A Real Cyberattack Event Network

To evaluate CERF, we use a GT CEN with nine months of actual attack events against a large US corporation. The GT is provided by the Cyberattack Automated Unconventional Sensor Environment (CAUSE) project,[2] and is available through agreement with the US Intelligence Advanced Research Projects Activity (IARPA). The GT focuses on significant attacks that current network defenses do not stop; it excludes spam email, routine network scans and other low-impact events. We filter GT further to events in which each label appears at least twenty times in the nine months. Although this GT is not openly available, it is representative of attacks against similar organizations in the same time period. It would be reasonable to extend our results using a data set for a similar organization.

As described in Section 7.5, the *event* predicate has $K + 1$ arguments. The

[2] https://www.iarpa.gov/index.php/research-programs/cause

first is time step T identifying the date. We list the remaining K roles, their logical variable names, and their label sets in Table 7.2. Role 1 identifies the class of attack: *phishing* events are emails containing malicious attachments or links, and *malware* events are malicious applications discovered on computer hosts. Roles 2-4 give details about the victim of the attack. We represent each GT CEN event as a grounding of event(T, C, J, L, G) with truth value 1. Similarities over labels are simple predefined measures of similarity (see Section 7.5.3.2), e.g., $\sigma_3(\texttt{MD}, \texttt{VA}) = .75$.

7.6.1.2 Cross Validation

We split the nine months of GT $M = m_1, \ldots, m_9$ into three sets (f_1, f_2, f_3) by month: $f_i = \left\{m_j \in M \mid (j-1) \bmod 3 + 1 = i\right\}$. We use the three sets for training sensors, training CERF, and testing CERF, respectively. We define the sets this way so each is representative of the entire period, to limit over-fitting caused by correlations across sets or caused by training CERF on the same events used to train its inputs (the sensors), and to have contiguous periods to train and test Rule (ER).

7.6.1.3 Sensor Graphs

We use one SG for each of the $K = 4$ roles. For each SG, we represent an edge between label R and time T with $\mathrm{s}_i(T, R)$. For example, the truth value of ground atom $\mathrm{s}_2(\texttt{Jan1}, \texttt{Acquisitions})$ represents our confidence that employees with job category `Acquisitions` will be attacked on `Jan1`. To generate the SGs, we train discriminative, multi-output classifiers on projections of f_1, conditioned on features extracted from the following original data sources: Twitter; the Global Database of Events, Language, and Tone (GDELT)[3] [101]; Open Threat Exchange; Wikidata; and Global Vectors for Word Representation (GloVe) [102]. The sensors produce SG outputs for day $T = t$ using evidence from $t - 7$, i.e., seven-day forecasts. CERF uses any time step given in a SG, whether past, present, or future. We refer to Okutan et al. for further details about how we use these sources [88, 103].

7.6.2 Systems

Our final CERF implementation used in the evaluation uses a subset of the possible partially-ground rules: We ground Rule (EP) on the following pairs of terms: (T, L), (C, L), (J, G), and (L, G), which we found balances accuracy and efficiency. We use day of week as categories for dates and treat all other labels as singleton categories. The final model has 518 rules with weights trained on f_2.

We compare CERF with a baseline assuming roles are distributed independently within events. It uses Rule (RP), plus the simple prior, and we train it on f_2. Due to the sparseness of the t-norm used in PSL conjunctions, we actually do this join outside PSL as a normal product, then load the results into PSL.

7.6.3 AuROC and Lift

We calculate AuROC by comparing soft truth values from CERF and the baseline with true events in f_3. AuROC is .76 for the baseline, and .78 for CERF, confirming CERF's over-all improvement to accuracy, but we are especially interested

[3] https://www.gdeltproject.org

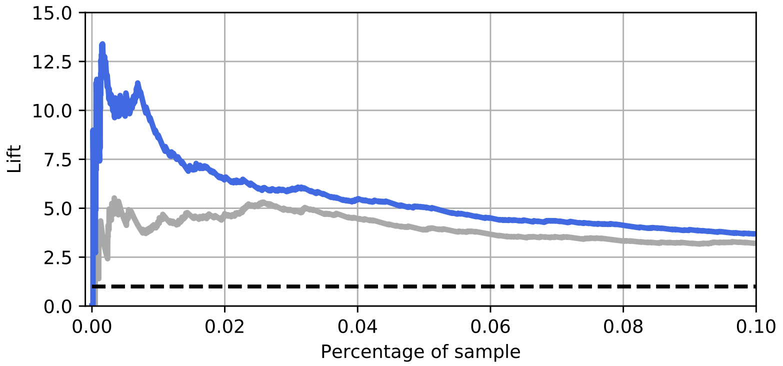


Figure 7.9: Lift of CERF (blue, top) and the baseline (grey).

in the rate of false positives among high-confidence predictions, as analysts are only able to review a small number of predictions every day. Lift is a better measure than AuROC for this. In Figure 7.9 we show the lift of CERF (blue, top) compared to the baseline (grey). Applying a threshold on truth value predicts a sample of the whole population of some size $\leq nu$ (horizontal axis of Figure 7.9). The lift of a detector is the ratio of its precision in that sample to expected precision (percent of the total population that is positive). For high-confidence predictions, CERF has up to double the lift of the baseline, indicating that analysts using CERF recommendations will have fewer false positives, compared to using sensors alone.

7.6.4 Partially-Observed Events

Analysts learn of some attacks as they occur. To measure our ability to leverage these *partially-observed* events, we remove a random sample of events from test set f_3 (so later accuracy measurements ignore them) and add them to the inputs of each model with truth value 1.0 (see Section 7.5.3), without retraining the models. In Figure 7.10, we measure AuROC for both systems as we vary the size (as a percentage

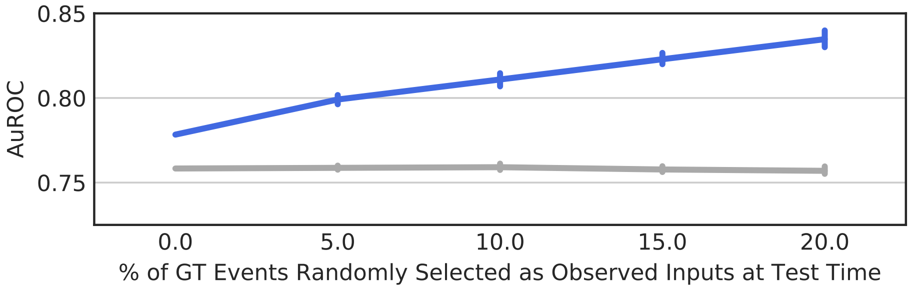


Figure 7.10: AuROC (95% CI) of CERF (blue, top) and the baseline (grey) as partial observation varies from zero to 20%.

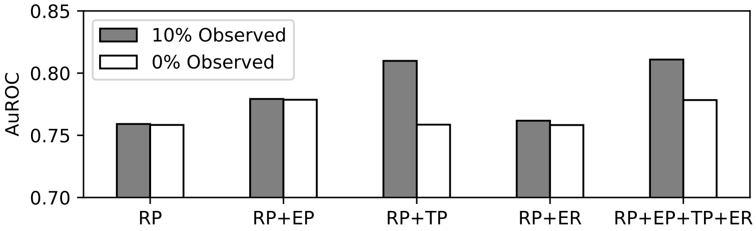


Figure 7.11: AuROC of CERF (RP+EP+TP+ER), the baseline Rule (RP), and rules EP, TP, and ER.

of f_3) of the random sample. For each setting, we repeat with five different random sets of observed events. For zero observed events, we do not measure repeatedly as results are constant. The baseline lacks rules crossing events, so its accuracy changes little with observed events. CERF uses its event-crossing rules (TP, ER) to increase accuracy with the size of the observed set. This suggests that when some attacks are known, CERF can rapidly alert analysts to other likely attacks.

7.6.5 Rule Contributions

Rule (RP) is the baseline and necessary to make predictions; without it, the rules could be trivially satisfied by predicting no events. To understand the contributions of the remaining three rules, we enable each in combination with RP. We use two settings for partially-observed events: one at 10% (results averaged over five random sets), and another with 0%. We plot AuROC in Figure 7.11. Compared to the baseline, Rule (EP) is more accurate, but it is insensitive to observed events – as expected, because EP focuses on event composition. For both cross-event rules (TP and ER), without observed events their accuracy is similar to the baseline. This suggests that the benefits of the current similarity measures are small when no events are observed. However, with observed events their accuracy increases substantially – especially with TP. CERF (RP+EP+TP+ER) combines all four types of rules and inherits their strengths.

7.6.6 Scalability

Increasing K, u, or n increases the cost of solving Equation (7.1) (see Section 7.4.4), but for the GT CEN with $K = 4$, $u = 5600$, and $n = 269$ (over 1.5M RVs), CERF learns the model and infers events in just 69 and 10 minutes, respectively. Cost also varies by which of the $p = 518$ rules are enabled. In Figure 7.12 we plot running times with different subsets of rules. Rule (EP) contributes the longest running time because it acts as a template with more partial-groundings than the other rules (Section 7.5.4.1). We ran all experiments on a machine with 16 2.67GHz Xeon cores and 256GB RAM.

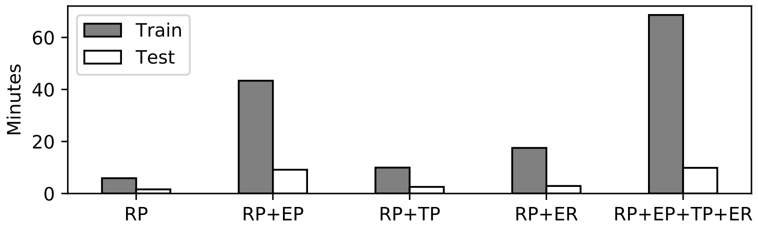


Figure 7.12: Running time of CERF (RP+EP+TP+ER), the baseline Rule (RP), and the three remaining rules.

7.7 Conclusion

Fusing multiple relational sources is a challenge that arises in data integration, yet conventional transformations are ill-suited for the problem. We introduce the new event-relational model to solve the source fusion and structured prediction problems. We apply the model to the problem of predicting cyberattack event networks. We evaluate CERF, the resulting system, on a CEN with nine months of real cyberattacks against a US corporation. We confirm CERF predicts events more accurately than sensors alone. Results suggest high utility for cybersecurity analysts, especially if some actual attacks are already known. We anticipate expanding or refining the sensors, similarity functions, and categories used in the model would increase accuracy further.

Chapter 8: Conclusion and Future Work

In this dissertation, I introduce new relational data integration approaches employing collective, probabilistic reasoning to handle inputs that can be diverse, noisy, and ambiguous. I focus on the problem of learning the heterogeneous structures of data sources – a necessary step before many data science tasks. The difficulty and urgency of this task is growing due the vast number of sources, the rapidness of changes to sources, the complexity of sources, and the richness and variety of target applications to which sources must be mapped.

I employ state-of-the-art techniques from two fields: (a) from the data integration field, I use well-developed methods for schema matching, mapping, and data exchange; and (b) from the machine learning field, I use a recently-developed framework for probabilistic learning – probabilistic soft logic (PSL) – which has the scalability and representational power to work on demanding data integration problems. As key contributions, I introduce new approaches to seven specific challenges in relational data integration:

1. *Combining metadata and data* to mitigate noise in metadata

2. *Probabilistic inference* for handling noise in data examples

3. *Homomorphism-based optimization* to handle partial outputs correctly

4. *Collective, prioritized disjunction rules* to handle ambiguity in data examples

5. *Boosted search* to correct flaws in transformations derived from metadata

6. *Joint matching and mapping* allowing uncertainty in correspondences

7. *Statistical transformations* for problems involving multi-source fusion

To evaluate the new approaches, I use an extensive and novel set of generated data integration problems. I also demonstrate the utility of the approaches on a variety of important, real data domains.

8.1 Open Challenges and Future Work

The broader problem of relational data integration is still far from solved. In the following sections, I discuss five open challenges for relational data integration, specific limitations in my contributed approaches, and promising directions for future work to address those limitations.

8.1.1 Input Types

In Chapter 3, I show how to combine a data example with metadata to select a mapping optimally. A data example is possible when source and target schemas share the same subject domain and we have instances for both schemas describing the same objects and events in that domain. These types of instances can be constructed or found occurring naturally, e.g., structured data on the web [104]. A related approach in the machine translation field is the use of parallel or strongly comparable corpora [105, 106]. In Chapter 6 I show how to select a mapping by reasoning jointly with metadata matches, which does not require the same kind of comparable data. Combining the approaches in Chapters 3 and 6 would be possible by combining their sets of PSL rules, and could be a basis for extending the benefits of both approaches to data instances that are a mixture of strongly and weakly comparable.

I focus on relational data because it is common in data management and analysis systems. Nested data representations are also common – especially for data interchange. Extending the mapping approaches in Chapters 3 and 5 for nested schemas would increase the number and variety of sources that could be used. To do this, a key problem is to create new refinement operators (see Section 5.4) for nested data. The mapping objectives should also be adapted; fortunately, many of the underlying algorithms – such as the chase – have already been adapted by others to nested data.

8.1.2 Transformation Languages

In Chapters 3, 5, and 6, I use st tgd mappings, which allow rich constraints over source data that should be transformed to the target instance. Putting the same st tgd logical variable in two arguments of the same relation, or in multiple relations, restricts it to the source tuples satisfying those constraints. Learning st tgds with the correct constraints avoids errors in target solutions. Even finer-grained source constraints are possible if we extend the mapping objective and refinement operators for mappings constraining variables to specific constant values and using constraints other than equality, such as arithmetic relations [33].

The st tgd language can also be extended to use custom functions in arguments of target relations. The st tgd mappings I use can generate target tuples with constants or with labeled nulls. A custom function is more flexible because it can apply a transformation on a constant before placing it in a target tuple, or combine multiple source constants into a new constant in place of a labeled null [13]. Adapting the mapping objectives from Chapters 3 and 5 for custom functions, or for additional source constraints, is made easier because – as with nested representations – many of the underlying algorithms have already been adapted by others for mappings with those features. A more interesting problem is to adapt refinement operators to generate refinements with those functions, and to incorporate new heuristics into the refinement process to handle the larger space of mappings with the extensions.

In Chapter 7 I demonstrate using statistical transformations in the probabilistic logical language PSL instead of logical st tgds, which have discrete truth values. A natural extension of this work is to adapt data exchange systems to use mappings with both kinds of rules.

8.1.3 Search

The search approach in Chapter 5 uses refinement operators to explore the space of possible st tgd mappings. With repeated applications of refinement operators, the number of candidate st tgds grows, but avoids an exponential blow up through operator design and validity checks. However, optimizing running times for search over very large schemas could make the approach even more practical. There are a variety of common search techniques that could be used for this. For example, refinement operators can consider type information, foreign keys, and correspondences when selecting variables to join.

The search approach running time can also be optimized in the step in which we calculate the mapping objective for a specific set of st tgds. That step, which includes queries over the source and target data instances, is currently the most expensive step in the search approach. A number of simple optimizations could significantly speed it up. For example, indexes over data instances could speed up queries. Also, boosting focuses the search on successively smaller subsets of the target instance in each stage, and the queries could be optimized to run only over that subset.

The search accommodates a wide variety of inputs, and handles flaws in st tgds; this is especially true for flaws causing errors that could be fixed by adding a constraint to a st tgd, e.g., joining two variables (see Section 5.4.1). However, errors caused by st tgds that are already over-constrained are more challenging to fix. For example, if an expert-provided st tgd already incorrectly places source data into the wrong target relation argument, that flaw prevents the st tgd from having any homomorphisms to the target instance. Searching for which argument is incorrect is inefficient. An interesting problem is to develop a way of measuring errors that guides the search to the individual arguments with incorrect variables, similar to the way we already guide the search to explain arguments of tuples.

8.1.4 External Interaction

In the mapping approaches from Chapters 3 and 5, expert users can improve mappings by suggesting candidate st tgds and by providing data examples. Other forms of user interaction may be easier or more effective. For example, some approaches allow users to incrementally refine data examples with feedback from the mapping system [18], and others allow users to annotate tuples [107]. New approaches combining those user interaction methods with the mapping approaches in this dissertation could combine the benefits of both and improve the practical benefits for users.

8.1.5 Resources

In Chapters 3 and 6, I use iBench [43, 76] – a system for generating diverse and realistic integration scenarios – to evaluate our mapping approaches. In Chapter 5, I use a novel set of generated mapping scenarios with parameters for controlling the search problem difficulty. Integrating the new scenarios as primitives in a full featured scenario generation system, like iBench, would have the benefit of generating even more realistic and challenging scenarios. It would also help to encourage reuse of problem sets within the community for effective comparisons of integration systems.

Evaluation on generated integration scenarios ensures an evaluation has good coverage over important mapping types with varying levels of difficulty. It is equally important to evaluate the same approaches on a variety of real data problems, such as the Amalgam and Neuroscience data sets used in Chapters 3, 5, and 6.

Expanding the set of publicly available benchmark integration data sets is important to improve mapping approaches and for effective comparisons of integration systems. It is especially important to share data sets having data instances for both the source and target, where those instances can be strongly comparable, weakly comparable, or a mixture of the two.

To summarize, while the broader problem of relational data integration is far from solved, my dissertation has presented new approaches for seven key challenges in this important problem area. While there is much more work to be done, I have shown how collective, probabilistic reasoning can help address these challenges by handling inputs that are diverse, noisy, and ambiguous.

Bibliography

[1] X. L. Dong and D. Srivastava. Big data integration. *Synthesis Lectures on Data Management*, 7(1):1–198, February 2015. ISSN: 2153-5418. DOI: `10.2200/S00578ED1V01Y201404DTM040`.

[2] M. J. Cafarella, A. Halevy, and J. Madhavan. Structured data on the web. *Communications of the ACM*, 54(2):72–79, 2011.

[3] Y. Velegrakis, R. J. Miller, and L. Popa. Mapping adaptation under evolving schemas. In *Proceedings of the VLDB Endowment (PVLDB)*, pages 584–595. VLDB Endowment, 2003.

[4] M. Q. Stearns, C. Price, K. A. Spackman, and A. Y. Wang. SNOMED clinical terms: overview of the development process and project status. In *Proceedings of the AMIA Symposium*, page 662. American Medical Informatics Association, 2001.

[5] R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema mapping as query discovery. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2000.

[6] S. H. Bach, M. Broecheler, B. Huang, and L. Getoor. Hinge-loss Markov random fields and probabilistic soft logic. *Journal of Machine Learning Research (JMLR)*, 18:1–67, 2017.

[7] A. Kimmig, A. Memory, R. J. Miller, and L. Getoor. A collective, probabilistic approach to schema mapping. In *IEEE Proceedings of the International Conference on Data Engineering (ICDE)*, 2017.

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

[9] A. Kimmig, A. Memory, R. J. Miller, and L. Getoor. A collective, probabilistic approach to schema mapping using diverse noisy evidence. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2018.

[10] A. Memory and W. G. Mueller. Sensor fusion and structured prediction for cyberattack event networks. In *15th International Workshop on Mining and Learning with Graphs (MLG)*, 2019.

[11] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[12] H. H. Do and E. Rahm. COMA: a system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2002.

[13] R. Dhamanka, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: discovering complex semantic matches between database schemas. In *ACM SIGMOD International Conference on the Management of Data*, pages 383–394, 2004.

[14] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: probabilistic alignment of relations, instances, and schema. *Proceedings of the VLDB Endowment (PVLDB)*, 5(3):157–168, 2011.

[15] W. Hu, J. Chen, H. Zhang, and Y. Qu. Learning complex mappings between ontologies. In *The Semantic Web*, pages 350–357. Springer, 2012.

[16] R. J. Miller, M. A. Hernandez, L. M. Haas, L. L. Yan, C. T. H. Ho, R. Fagin, and L. Popa. The Clio project: managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[17] A. Bonifati, G. Mecca, P. Papotti, and Y. Velegrakis. Discovery and correctness of schema mapping transformations. *Schema Matching and Mapping*:111–147, 2011.

[18] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Designing and refining schema mappings via data examples. In *ACM SIGMOD International Conference on the Management of Data*, 2011.

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

[20] C. Zhang, R. Hoffmann, and D. S. Weld. Ontological smoothing for relation extraction with minimal supervision. In *AAAI Conference on Artificial Intelligence*, 2012.

[21] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 598–609, August 2002.

[22] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger. HePToX: marrying XML and heterogeneity in your P2P databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2005.

[23] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. ++Spicy: an open source tool for second-generation schema mapping and data exchange. *Proceedings of the VLDB Endowment (PVLDB)*, 4(12):1438–1441, 2011.

[24] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema mapping verification: the Spicy way. In *International Conference on Extending Database Technology (EDBT)*, pages 85–96, 2008.

[25] J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *ACM SIGMOD International Conference on the Management of Data*, pages 205–216, 2003.

[26] L.-L. Yan, R. J. Miller, L. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. *SIGMOD Record*, 30(2):485–496, 2001.

[27] B. Alexe, L. Chiticariu, R. J. Miller, and W.-C. Tan. Muse: mapping understanding and design by example. In *IEEE Proceedings of the International Conference on Data Engineering (ICDE)*, 2008.

[28] B. ten Cate, P. G. Kolaitis, and W. C. Tan. Schema mappings and data examples. In *International Conference on Extending Database Technology (EDBT)*, 2013.

[29] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. EIRENE: interactive design and refinement of schema mappings via data examples. *Proceedings of the VLDB Endowment (PVLDB)*, 4(12):1414–1417, 2011.

[30] B. ten Cate, V. Dalmau, and P. G. Kolaitis. Learning schema mappings. *ACM Transactions on Database Systems (TODS)*, 38(4):28, 2013.

[31] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *Proceedings of the International Conference on Database Theory (ICDT)*, 2010.

[32] G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *Journal of the ACM (JACM)*, 57(2):6, 2010.

[33] B. ten Cate, P. G. Kolaitis, K. Qian, and W.-C. Tan. Approximation algorithms for schema-mapping discovery from data examples. In *Proceedings of the 9th Alberto Mendelzon International Workshop on Foundations of Data Management*, page 24, 2015.

[34] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 61–75, 2005.

[35] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Elsevier, 2012.

[36] M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Foundations of schema mapping management. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 227–238. ACM, 2010.

[37] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Reverse data exchange: coping with nulls. *ACM Transactions on Database Systems (TODS)*, 36(2):11, 2011.

[38] P. C. Arocena, B. Glavic, and R. J. Miller. Value invention in data exchange. In *ACM SIGMOD International Conference on the Management of Data*, pages 157–168. ACM, 2013.

[39] G. Mecca, P. Papotti, and S. Raunich. Core schema mappings. In *ACM SIGMOD International Conference on the Management of Data*, pages 655–668. ACM, 2009.

[40] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *Proceedings of the VLDB Endowment (PVLDB)*, 6(9):625–636, 2013.

[41] Z. Bellahsene, A. Bonifati, F. Duchateau, and Y. Velegrakis. On evaluating schema matching and mapping. In *Schema matching and mapping*, pages 253–291. Springer, 2011.

[42] G. Mecca, P. Papotti, S. Raunich, and D. Santoro. What is the IQ of your data transformation system? In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 872–881, 2012.

[43] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller. The iBench integration metadata generator. *Proceedings of the VLDB Endowment (PVLDB)*, 9(3):108–119, 2015.

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

[45] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[46] H. Poon and P. Domingos. Joint unsupervised coreference resolution with Markov logic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 650–659. Association for Computational Linguistics, 2008.

[47] M. Niepert, C. Meilicke, and H. Stuckenschmidt. A probabilistic-logical framework for ontology matching. In *AAAI Conference on Artificial Intelligence*, 2010.

[48] J. Pujara, H. Miao, L. Getoor, and W. W. Cohen. Using semantics and statistics to turn data into knowledge. *AI Magazine*, 36(1):65–74, 2015.

[49] L. De Raedt. *Logical and Relational Learning*. Springer-Verlag New York Inc, 2008.

[50] J. Lee, R. Marinescu, and R. Dechter. Applying marginal MAP search to probabilistic conformant planning: initial results. In *AAAI Workshop: Statistical Relational Artificial Intelligence*, 2014.

[51] J. C. Beck and M. S. Fox. A generic framework for constraint-directed search and scheduling. *AI Magazine*, 19(4):103, 1998.

[52] S. H. Bach, B. Huang, and L. Getoor. Unifying local consistency and MAX SAT relaxations for scalable inference with rounding guarantees. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 46–55, 2015.

[53] S. H. Bach, B. Huang, B. London, and L. Getoor. Hinge-loss Markov random fields: convex inference for structured prediction. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, page 32, 2013.

[54] B. ten Cate and P. G. Kolaitis. Structural characterizations of schema-mapping languages. *Communications of the ACM*, 53(1):101–110, 2010.

[55] H. Elmeleegy, A. K. Elmagarmid, and J. Lee. Leveraging query logs for schema mapping generation in U-MAP. In *ACM SIGMOD International Conference on the Management of Data*, 2011.

[56] L. Qian, M. J. Cafarella, and H. Jagadish. Sample-driven schema mapping. In *ACM SIGMOD International Conference on the Management of Data*, 2012.

[57] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Characterizing schema mappings via data examples. *ACM Transactions on Database Systems (TODS)*, 36(4):23, 2011.

[58] J. Pujara, H. Miao, L. Getoor, and W. Cohen. Knowledge graph identification. In *International Semantic Web Conference (ISWC)*, 2013.

[59] S. Fakhraei, B. Huang, L. Raschid, and L. Getoor. Network-based drug-target interaction prediction with probabilistic soft logic. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 11(5):775–787, 2014.

[60] P. Kouki, S. Fakhraei, J. Foulds, M. Eirinaki, and L. Getoor. HyPER: a flexible and extensible probabilistic framework for hybrid recommender systems. In *Proceedings of the ACM Conference on Recommender Systems (RecSys)*, pages 99–106. ACM, 2015.

[61] G. Mecca, P. Papotti, and D. Santoro. IQ-METER - an evaluation tool for data-transformation systems. In *IEEE Proceedings of the International Conference on Data Engineering (ICDE)*, 2014.

[62] R. Fagin, L. M. Haas, M. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236. Springer, 2009.

[63] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

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

[65] D. Barbosa, A. O. Mendelzon, J. Keenleyside, and K. A. Lyons. ToXgene: a template-based data generator for XML. In *ACM SIGMOD International Conference on the Management of Data*, 2002.

[66] R. J. Miller, D. Fisla, M. Huang, D. Kymlicka, F. Ku, and V. Lee. The Amalgam schema and data integration test suite, 2001.

[67] C. Becker, C. Bizer, M. Erdmann, and M. Greaves. Extending SMW+ with a linked data integration framework. In *International Semantic Web Conference (ISWC)*, 2011.

[68] C. Knoblock, P. Szekely, J. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, and P. Mallick. Semi-automatically mapping structured sources into the semantic web. In *The Semantic Web: Research and Applications*. Volume 7295, Lecture Notes in Computer Science, pages 375–390. Springer, 2012. ISBN: 978-3-642-30283-1.

[69] C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[70] A. Gal. Managing uncertainty in schema matching with top-K schema mappings. *Journal on Data Semantics VI: Special Issue on Emergent Semantics*:90–114, 2006.

[71] 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 *IEEE Proceedings of the International Conference on Data Engineering (ICDE)*, 2014.

[72] 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. *Proceedings of the VLDB Endowment (PVLDB)*, 6(3):205–216, 2013.

[73] X. L. Dong, B. Saha, and D. Srivastava. Less is more: selecting sources wisely for integration. *Proceedings of the VLDB Endowment (PVLDB)*, 6(2):37–48, 2012.

[74] L. Badea. A refinement operator for theories. In *International Conference on Inductive Logic Programming*, pages 1–14. Springer, 2001.

[75] B. L. Richards and R. J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

[76] B. Alexe, W.-C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1):230–244, 2008.

[77] L. Mihalkova and R. J. Mooney. Bottom-up learning of Markov logic network structure. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 625–632. ACM, 2007.

[78] S. Kok and P. Domingos. Learning Markov logic networks using structural motifs. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 551–558, 2010.

[79] T. N. Huynh and R. J. Mooney. Online structure learning for Markov logic networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 81–96. Springer, 2011.

[80] S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 441–448, 2005.

[81] T. Khot, S. Natarajan, K. Kersting, and J. Shavlik. Learning Markov logic networks via functional gradient boosting. In *2011 IEEE 11th International Conference on Data Mining (ICDM)*, pages 320–329. IEEE, 2011.

[82] M. Biba, S. Ferilli, and F. Esposito. Discriminative structure learning of Markov logic networks. In *International Conference on Inductive Logic Programming*, pages 59–76. Springer, 2008.

[83] S. H. Bach, M. Broecheler, L. Getoor, and P. D. O'Leary. Scaling MPE inference for constrained continuous Markov random fields with consensus optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[84] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems (TODS)*, 4(4):455–469, 1979.

[85] A. Cali, D. Calvanese, G. De Giacomo, and M. Lenzerini. On the expressive power of data integration systems. In *International Conference on Conceptual Modeling*, pages 338–350. Springer, 2002.

[86] Ponemon Institute. 2018 Cost of a Data Breach Study: Global Overview. Technical report, Ponemon Institute LLC, 2018.

[87] S. Zong, A. Ritter, G. Mueller, and E. Wright. Analyzing the perceived severity of cybersecurity threats reported on social media. *arXiv preprint arXiv:1902.10680*, 2019.

[88] A. Okutan, S. J. Yang, and K. McConky. Predicting cyber attacks with Bayesian networks using unconventional signals. In *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, page 13. ACM, 2017.

[89] A. Dalton, B. Dorr, L. Liang, and K. Hollingshead. Improving cyber-attack predictions through information foraging. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4642–4647. IEEE, 2017.

[90] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 282–289, San Francisco, CA, USA, 2001.

[91] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6(Sep):1453–1484, 2005.

[92] F. Mirzazadeh, S. Ravanbakhsh, N. Ding, and D. Schuurmans. Embedding inference for structured multilabel prediction. In *Advances in Neural Information Processing Systems*, pages 3555–3563, 2015.

[93] S. Riedel, H.-W. Chun, T. Takagi, and J. Tsujii. A Markov logic approach to bio-molecular event extraction. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing: Shared Task*, pages 41–49. Association for Computational Linguistics, 2009.

[94] A. Memory, A. Kimmig, S. H. Bach, L. Raschid, and L. Getoor. Graph summarization in annotated data using probabilistic soft logic. In *Proceedings of the 8th International Conference on Uncertainty Reasoning for the Semantic Web-Volume 900*, pages 75–86. CEUR-WS, 2012.

[95] Q. Liu, S. Wu, and L. Wang. Collaborative prediction for multi-entity interaction with hierarchical representation. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 613–622. ACM, 2015.

[96] S. Watanabe. Information theoretical analysis of multivariate correlation. *IBM Journal of research and development*, 4(1):66–82, 1960.

[97] L. A. Goodman. Snowball sampling. *The annals of mathematical statistics*, 32(1):148–170, 1961.

[98] P. Hu and W. C. Lau. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865*, 2013.

[99] K. Zhang and K. Pelechrinis. Understanding spatial homophily: the case of peer influence and social selection. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 271–282, New York, NY, USA. ACM, 2014.

[100] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[101] K. Leetaru and P. A. Schrodt. GDELT: global data on events, location, and tone. *ISA Annual Convention*, 2013.

[102] J. Pennington, R. Socher, and C. D. Manning. GloVe: global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[103] A. Okutan, S. J. Yang, and K. McConky. Forecasting cyber attacks with imbalanced data sets and different time granularities. *arXiv preprint arXiv: 1803.09560*, 2018.

[104] N. Dalvi, A. Machanavajjhala, and B. Pang. An analysis of structured data on the web. *Proceedings of the VLDB Endowment (PVLDB)*, 5(7):680–691, 2012.

[105] D. S. Munteanu and D. Marcu. Improving machine translation performance by exploiting non-parallel corpora. *Computational Linguistics*, 31(4):477–504, 2005.

[106] J. R. Smith, C. Quirk, and K. Toutanova. Extracting parallel sentences from comparable corpora using document level alignment. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 403–411. Association for Computational Linguistics, 2010.

[107] K. Belhajjame, N. W. Paton, S. M. Embury, A. A. A. Fernandes, and C. Hedeler. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *International Conference on Extending Database Technology (EDBT)*, pages 573–584, 2010.