UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DISCOVERING INFORMATION INTEGRATION SPECIFICATIONS FROM DATA EXAMPLES

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Kun Qian

March 2017

The Dissertation of Kun Qian
is approved:

Table of Contents

List of Figures

List of Tables

Abstract

Discovering Information Integration Specifications from Data Examples

by

Kun Qian

Two fundamental problems in information integration are data exchange and entity resolution. Data exchange is the task of translating data structured under a source schema into data structured under a target schema. Data exchange is captured by schema mappings that specify the relationship between a source schema and a target schema at a high level. Entity resolution is the task of identifying and linking different representations of the same real-world object. The goal of entity resolution is to create links among existing data. Although schema mapping and entity resolution have been successfully used in many domains, manually designing schema mappings and entity resolution algorithms is a labor-intensive and time-consuming process.

In this dissertation, we develop example-driven discovery/learning methods for high-level declarative schema mapping specifications and high-level declarative entity resolution algorithms. This dissertation contains two parts. In Part I, we present our work on extending and refining two major example-driven schema-mapping discovery frameworks, namely, the repair framework introduced by Gottlob and Senellart and the learning framework introduced by ten Cate et al. Gottlob and Senellart introduced a framework for schema-mapping discovery from a single data example, in which the derivation of a schema mapping is cast as an optimization problem. We refine and study this framework in more depth. Among other results, we design a polynomial-time $\log(n)$-approximation algorithm for computing optimal schema mappings from a given set of data examples for a restricted class of schema mappings; moreover, we show that this approximation ratio cannot be improved. We implemented the aforementioned $\log(n)$-approximation algorithm and carried out an experimental evaluation in a real-world mapping scenario. As opposed to the repair framework, in which the schema-mapping discovery problem is cast as an optimization problem, the derivation of a schema mapping is cast as a computational learning problem in the learning framework. We design a learning algorithm that is an Occam algorithm leading up to a PAC learning algorithm for an important class of schema mappings. We also implemented the proposed algorithm and carried out an experimental evaluation using mapping scenarios created by iBench, which is a state-of-the-art benchmarking tool. In Part II, we introduce a new active learning system for entity resolution that learns high-quality entity resolution algorithms. Our focus is on learning entity resolution algorithms in big data scenarios. We implemented the aforementioned active learning system and carried out an experimental evaluation in two real-world big data entity resolution scenarios.

Acknowledgments

I am extremely grateful to my three advisors Balder ten Cate, Phokion Kolaitis, and Wang-Chiew Tan for their constant support, patience and generosity during my time as a PhD student in the database group at UC Santa Cruz. Phokion is not only a world-class researcher but also an excellent teacher who is extremely knowledgable in the computational theory literature. Wang-Chiew is also an excellent adviser who always offers me constructive and insightful suggestions. Balder is remarkably brilliant and very kind. He taught me a lot about research and I greatly appreciate the long emails (which included detailed comments to my questions) he has sent me over the past four years. It is truly a privilege for me to work with these three outstanding researchers. I am also thankful to Peter Alvaro, who is fourth member of my dissertation reading committee, for his enthusiasm in reading for this dissertation.

During my PhD career, I had two summer internships at IBM Research Almaden. I am very thankful to my mentors at IBM: Bogdan Alexe, Mauricio Hernandez, Lucian Popa, and Prithviraj Sen. In particular I would like to thank Lucian who encouraged me to apply for a full-time position at IBM Research Almaden. Without his support, I would have not received an offer from IBM.

A special word of thanks also goes to my family for their continuous support and encouragement.

Chapter 1

Introduction

Information integration is the task of merging and combining information from heterogeneous sources with different representations into a unified view of information that is meaningful and valuable for the subsequent data analytic applications. Two fundamental problems in information integration are *data exchange* and *entity resolution (or, ER in short)*. Data exchange is the task of translating data structured under a source schema into data structured under a target schema. Entity resolution, also known as several other names (record matching, deduplication, entity matching, etc.), is the task of identifying and linking different representations of the same real-world object. As opposed to data exchange, the goal of which is to create a target instance from a source instance, the goal of entity resolution is to find *matches* among existing data.

Declarative Approaches to Data Exchange and Entity Resolution

Syntactically, both data exchange and entity resolution can be expressed in a high-level, declarative fashion. In data exchange, schema mappings are used to specify the relationships between the schemas involved. Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ be a target schema. A schema mapping is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a collection of high-level declarative assertions that specify the relationship between $\mathbf{S}$ and $\mathbf{T}$. It follows that, given an instance I that conforms to $\mathbf{S}$ and given a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, the goal of data exchange is to create an instance J over $\mathbf{T}$ using $\mathcal{M}$ such that $\mathcal{M}$ is satisfied. Schema mappings are specified in some schema mapping language, such as the language of GLAV (Global-and-Local-As-View) constraints (also known as source-to-target tuple-generating dependencies, or s-t tgds). GLAV constraints belong to the universal-existential fragment of first-order logic. The language of GLAV constraints is considered to be a standard language for specifying schema mappings [23,29]. As an illustration, the following first-order sentence is an example of a GLAV constraint over a source schema $\mathbf{S} = \{\text{Student}(\texttt{studentID}),\ \text{Enrolls}(\texttt{studentID},\ \texttt{courseID})\}$ and a target schema $\mathbf{T} = \{\text{Teacher}(\texttt{teacherID},\ \texttt{courseID}),\ \text{Grade}(\texttt{studentID},\ \texttt{courseName},\ \texttt{grade})\}$:

$$\forall s \forall c\ \big(\ \text{Student}(s) \land \text{Enrolls}(s,c) \rightarrow \exists t \exists g\ \text{Teacher}(t,c) \land \text{Grade}(s,c,g)\ \big).$$

Semantically, the GLAV constraint above specifies that for every student s who enrolls in the course c, there is a teacher t who teaches the course c and the student s receives a grade g. Once a GLAV schema mapping $\mathcal{M}$ between $\mathbf{S}$ and $\mathbf{T}$ is established, we can then translate[1] an instance I over $\mathbf{S}$ to an instance J over $\mathbf{T}$ using $\mathcal{M}$. Schema-mapping design is therefore the task of deriving high-level declarative specifications (expressed in the language of GLAV constraints) that describe the relationship between a source schema and a target schema.

Entity resolution is typically considered to be an algorithmic problem. An entity resolution task has one or more input datasets. The problem of entity resolution is to find all pairs of records, among the input datasets, that represent the same real-world object. Both the research community and the industry have been working on developing different ER algorithms for various ER tasks (see [31, 35, 36]). Traditionally, ER algorithms are implemented as statistical pairwise classifiers that determine whether a pair of records is a match or a non-match. Since the late 1960s, various advanced machine learning techniques have been developed for ER tasks (e.g., [14, 26, 28, 32, 39]). These works have put significant efforts into designing, testing, and tuning ER algorithms that are based on programs written in a procedural fashion. Lately, there has been research that developed new frameworks that define the process of entity resolution in a logical formalism, such as [6, 7, 19, 34, 42]. In particular, the language of HIL [42], introduced by IBM, specifies the process of entity resolution in the language of *entity resolution rules* (or, *matching rules).* ER rules are SQL-like declarative constraints. Figure 1.1 shows an example of an ER rule (i.e., `Rule1`) that is used to detect matches between a set of Twitter user profiles and a set of customer records (e.g., from Walmart). Semantically, `Rule1` specifies that a Twitter user profile and a customer record are considered to be

[1]The translation can be done by performing the chase procedure introduced in [29].

a match if (1) they have the exact same last name, (2) their first names match via a user-defined function (i.e., `firstNameMatch(x,y)`), and (3) their states are the same.

```
match Twitter T, Customer C by Rule1 :
   T.lastname = C.lastname
   AND firstNameMatch(T.firstName, C.firstName)
   AND T.state = C.state
```

Figure 1.1: An example of an ER rule

Rule-based ER algorithms have several good properties including: (1) they are relatively easy to understand (rules are typically human-readable), (2) they are relatively easy to maintain and customize, and (3) they are amenable to efficient evaluation on datasets (e.g., `Rule1` could be evaluated as a regular database join between the Twitter and Customer datasets). The result of executing an ER rule `R` is a set of *links*, that is, pairs of records that satisfy `R`. A rule-based ER algorithm can include a disjunctive of rules. Therefore, the result of executing an rule-based ER algorithm A is the set of links obtained by taking the union of the links produced by each ER rule of A. The entity resolution learning problem is therefore the task of learning (rule-based) entity resolution algorithms that can be subsequently used to create links with high precision and high recall (see more details in Section 4.2 of Chapter 4).

Defining schema mappings and entity resolution in a high-level, declarative fashion allows the user to focus on the computation at a logical level without spelling out the implementation details relevant to the physical level. The above discussion briefly reviewed the language of GLAV constraints for schema mappings and the language of ER rules for entity resolution. Although they are two different languages for two different information integration problems, there is a connection between them. In Section 4.1 of Chapter 4, we discuss the similarities between the two languages and how they are related to each other.

Example-Driven Approaches to Schema Mapping Design and the Rule-Based Entity Resolution Algorithms

Many of today's data-centric applications rely on schema mappings and/or entity resolution. For instance, when a user searches for rental cars on some websites (e.g., Expedia.com), the user's query is sent to multiple rental car companies. Then a unified answer, which is obtained from multiple databases belonging to different rental car companies, is presented to the user. Schema mappings play important roles in such kind of application. Entity resolution also has been used in many tasks that are important in daily life. For instance, most credit report agencies use entity resolution to draw a complete picture of a person across multiple data sources. A second example is when two companies merge, they may need to combine their customer databases. If they have a customer in common, they need to create a new composite customer record that combines information from the two original records. Entity resolution is used to find the matching records in the two customer databases in question.

Although schema mapping and entity resolution have been successfully used in many domains, manually designing schema mappings and entity resolution algorithms is a labor-intensive and time-consuming process. Therefore, it is important to develop automatic or semi-automatic methods to facilitate the process of designing and refining schema mappings and entity resolution algorithms. In recent years, data examples have been at the core of many approaches both to schema-mapping design (e.g., [3, 21, 24, 38]) and to learning statistical entity resolution classifiers (see [36]). In the setting of schema-mapping discovery, a data example is a pair (I, J) consisting of a source instance I and a target instance J. In the setting of entity resolution learning, a data example is a pair (r, s) consisting of two records r and s, which can be labeled as a match or a non-match by a human user. In this dissertation, we develop example-driven discovery/learning methods for high-level declarative schema mapping specifications and high-level declarative rule-based entity resolution algorithms. In the rest of this chapter, we discuss the background of our work, limitations of existing offerings, and our contributions.

1.1 Discovering Schema Mapping Specifications from Data Examples

Deriving a schema mapping between two schemas can be an involved and time-consuming process. One reason is the complexity and scale of many real world schemas, which makes it difficult to understand which schema mapping should be derived and how they should be derived. Another reason is that, even if the source and target schemas are simple, there can still be many different, logically inequivalent, candidate schema mappings. In view of this state of affairs, several different approaches have been developed to facilitate the process of discovering the “correct” schema mapping between two schemas. Two main such approaches are the derivation of schema mappings through *graphic-interface systems* and the derivation of schema mappings via data examples.

Graphical-interface systems that are used to derive schema mappings first solicit a visual specification of correspondences between the elements of the two schemas from a user. In addition to depicting the source and target schemas, the interface allows the user to provide input by specifying which pairs of elements from the two schemas are related; moreover, a schema-matching module can also be used to derive such pairs. Once the visual specification (i.e., schemas and correspondences between elements of the schemas) has been completed, the system automatically produces a schema mapping. A shortcoming is that multiple logically inequivalent schema mappings may conform to the same visual specification [1], and these systems often make implicit assumptions about the visual specification to arrive at a single schema mapping. Examples of such systems include research prototypes (e.g., [16, 41, 56]) and commercial systems (e.g., Altova Mapforce [2] and Stylus Studio [3]).

In recent years, data examples have been at the core of several approaches to schema mapping design. A data example is a pair (I, J) consisting of an instance I conforming to a source schema and an instance J conforming to a target schema. Using data examples to facilitate the schema mapping design is a preferable way because, in a precise sense, data examples describe the underlying semantics between the source schema and the target schema in question. There are three main example-driven schema-mapping discovery frameworks:

[2]http://www.altova.com/mapforce.html

[3]http://www.stylusstudio.com/xml_mapper.html

- the *fitting* framework [3];
- the *Gottlob-Senellart* framework [38];
- the *learning* framework [21].;

The three example-driven frameworks draw from different areas, such as database, logic, and computational learning theory. We next provide a high-level review of the three frameworks.

- **Fitting framework.** In [3], the authors studied the problem that, given a set E of data examples, determines whether or not there is a GLAV schema mapping that *fits* E, and, if so, deriving such a schema mapping. Here, a schema mapping $\mathcal{M}$ *fits* a set E of data examples if for every data example (I, J) in E, we have that J is a universal solution of I w.r.t. $\mathcal{M}$, which, intuitively, means that J is a "most preferred" solution for I w.r.t. $\mathcal{M}$ [29]. A sound and complete algorithm for solving this problem was designed in [3] and experiments with its implementation were reported. Moreover, if, given a set of data examples, there exists a GLAV schema mapping that fits them, then the algorithm returns a fitting GLAV schema mapping $\mathcal{M}$ that is linear in the size of the data examples and has the property that it is the *most general* one, which means that if $\mathcal{M}'$ is some other fitting GLAV schema mapping, then $\mathcal{M}'$ logically implies $\mathcal{M}$.
- **Gottlob-Senellart framework.** In [38], the authors introduced and studied a cost model for deriving a schema mapping from a single *ground* data example, i.e., a data example consisting of instances without labelled nulls. Ideally, given a ground data example (I, J), one would like to find a GLAV schema mapping $\mathcal{M}$ that is

valid and *fully explaining*[4] for (I, J); here, *valid* means that (I, J) satisfies the constraints of $\mathcal{M}$, while *fully explaining* means that every fact in J belongs to every target instance K such that (I, K) satisfies the constraints of $\mathcal{M}$. However, such a schema mapping may not exist for a given data example. For this reason, Gottlob and Senellart developed a framework that uses two schema-mapping languages: the standard language of GLAV constraints and an extended language GLAV$^{=,\neq}$ of *repairs* that augments GLAV constraints with equalities, inequalities, and ground facts. The *cost* of a GLAV schema mapping $\mathcal{M}$ w.r.t. a data example (I, J) is the minimum size of a valid and fully explaining repair of $\mathcal{M}$, where a *repair* of $\mathcal{M}$ is a schema mapping $\mathcal{M}'$ in the extended language that can be obtained from $\mathcal{M}$ via a sequence of specified repair operations. Given a ground data example (I, J), the goal then is to find an *optimal* GLAV schema mapping for (I, J), that is to say, a GLAV schema mapping whose cost w.r.t. (I, J) is as small as possible.

- **Learning framework.** In [21], the derivation of a schema mapping is cast as a computational learning problem. In computational learning, the main task is to (exactly or approximately) identify a goal concept $\mathcal{G}$ by asking a number of queries about it through oracles. Typical queries are the following:
  - *Membership queries*, which ask the question that whether a data example e is a positive example for the goal concept $\mathcal{G}$. The answer is yes if e is positive for $\mathcal{G}$; otherwise, no.
  - *Equivalence queries*, which ask the question that whether a candidate concept

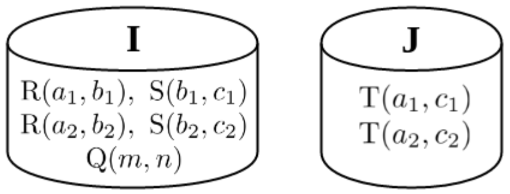
[4]In the setting of a ground data example, valid and fully explaining is equivalent to fitting

$\mathcal{H}$ is logically equivalent to $\mathcal{G}$. If yes, then the equivalence oracle returns true; otherwise, a counterexample e, which is positive example for one of the two concepts but not for both, is returned.

- *Random example queries*, which ask for randomly generated labeled examples according to the goal concept $\mathcal{G}$.

Under the learning framework introduced in [21], schema mappings are viewed as concepts which can be identified either by positive/negative examples or by *universal* examples. The learnablility of GAV (stands for Global-As-View) schema mappings were studied in the learning framework. The main question investigated in [21] is "*under what standard models of learning are GAV schema mappings learnable using data examples*?". Two well-known learning models were considered: (1) the *exact learning* model introduced by Angluin [5] and (2) the *PAC (Probably-Approximately-Correct)* learning model introduced by Valiant [69].

Although the three frameworks all derive schema mappings from data examples, there are essential differences among them. As an illustration, consider a simple schema-mapping discovery scenario, in which a single data example (I, J) (see below) is provided.



Assume that the intended schema mapping is $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where

$$\mathbf{S} = \{\mathrm{R}, \mathrm{S}, \mathrm{Q}\},\ \mathbf{T} = \{\mathrm{T}\},\ \text{and } \Sigma = \{\mathrm{R}(x, y) \wedge \mathrm{S}(y, z) \rightarrow \mathrm{T}(x, z)\}.$$

Here, the goal is to drive some schema mapping from (I, J) that is semantically close to the intended schema mapping $\mathcal{M}$.

When using the fitting approach, the fitting approach first checks whether there is a fitting schema mapping for (I, J) using a procedure called *homomorphism extension test* [3]. It turns out that the answer is yes, and thus the fitting approach returns the *canonical GLAV schema mapping* $\mathcal{M}_{fit} = (\mathbf{S}, \mathbf{T}, \Sigma_{fit})$, where Σ_{fit} (see below) is obtained by spelling out the facts in (I, J). By construction, $\mathcal{M}_{fit}$ is a fitting schema mapping for (I, J) that perfectly describes the transformation from I to J.

$$\Sigma_{fit} = \{\mathrm{R}(x_1, y_1) \wedge \mathrm{S}(y_1, z_1) \wedge \mathrm{R}(x_2, y_2) \wedge \mathrm{S}(y_2, z_2) \wedge \mathrm{Q}(u, v) \wedge \rightarrow \mathrm{T}(x_1, z_1) \wedge \mathrm{T}(x_2, z_2)\}.$$

When using the Gottlob-Senellart approach, the Gottlob-Senellart approach considers all schema mappings that are valid and fully explaining for (I, J), including $\mathcal{M}_{fit}$. However, the Gottlob-Senellart approach would not choose $\mathcal{M}_{fit}$ because there are other schema mappings that are "better" than $\mathcal{M}_{fit}$ according to its cost model. For example, consider the mapping $\mathcal{M}_{gs} = (\mathbf{S}, \mathbf{T}, \Sigma_{gs})$, where

$$\Sigma_{gs} = \{\mathrm{R}(x, y) \wedge \mathrm{S}(y, z) \rightarrow \mathrm{T}(x, z)\}.$$

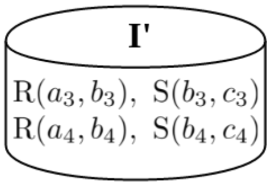
It can be verified that both $\mathcal{M}_{fit}$ and $\mathcal{M}_{gs}$ are valid and fully explaining for (I, J), but $\mathcal{M}_{gs}$ is syntactically more concise. In fact, according to the cost model of the Gottlob-Senellart framework, the mapping $\mathcal{M}_{gs}$ is considered to be an optimal solution for (I, J). We can see that $\mathcal{M}_{gs}$ is identical to the intended mapping $\mathcal{M}$.

When using the learning approach, the learning approach further assumes that two oracles (i.e., a membership oracle and an equivalence oracle) that can answer specific queries about $\mathcal{M}$ are given. Intuitively, in the setting of schema-mapping learning, a membership oracle can answer queries like *"is a particular data example a positive example for the goal schema mapping?"*; an equivalence oracle can answer queries like *"is a particular candidate schema mapping logically equivalent to the goal schema mapping?"*. When the two mappings are not logically equivalent, the equivalence oracle would also return a counterexample, that is, a data example that is positive for one of the two mappings but not for both. During the learning, the learning approach can use the provided oracles to generate a number of new data examples from (I, J). In the present schema-mapping discovery scenario, a new data example $(I^{\text{learn}}, J^{\text{learn}})$ would be generated from the input data example (I, J), where

$$I^{\text{learn}} = \{\text{R}(a_1, b_1), \text{S}(b_1, c_1)\} \textit{ and } J^{\text{learn}} = \{\text{T}(a_1, c_1)\}.$$

It follows that the learning approach would convert the data example $(I^{\text{learn}}, J^{\text{learn}})$ into a schema mapping that is identical to $\mathcal{M}_{gs}$.

Note that $\mathcal{M}_{fit}$ and $\mathcal{M}_{gs}$ are semantically indistinguishable with respect to (I, J). Now suppose that we use both $\mathcal{M}_{fit}$ and $\mathcal{M}_{gs}$ to translate a new source instance I' (see below) to a target instance.



It follows that $\mathcal{M}_{fit}$ would produce an empty target instance because the left-hand side of the GLAV constraint of $\mathcal{M}_{fit}$ is not triggered (i.e., the left-hand side of the constraint of $\mathcal{M}_{fit}$ is not satisfied in I') due to an absence of a corresponding Q-fact. In contrast, the mapping $\mathcal{M}_{gs}$ would produce the following instance J', which is the same instance that would be produced by applying $\mathcal{M}$ to I'.

$$J' = \{T(a_3, c_c), T(a_4, c_4)\}.$$

As shown in the schema-mapping discovery scenario above, the mapping produced by the fitting approach does not work well on the new instance I', and that is because the main focus of the fitting approach is on the fitting decision problem, namely, determining whether there is a fitting GLAV schema mapping for a finite set of data examples. If the answer is yes, then the fitting approach would return the canonical GLAV schema mapping that is very specific to the input example (I, J). In contrast, both the Gottlob-Senellart approach and the learning approach would not produce a mapping that is a full description of (I, J). In particular, the Gottlob-Senellart approach chooses the most syntactically succinct schema mapping among all valid and fully explaining schema mappings for (I, J). The learning approach avoid producing a full-description mapping of (I, J) by generating a number of guided data examples that would lead the learning algorithm towards a schema mapping that is semantically close to the intended schema mapping. However, the Gottlob-Senellart framework and the learning framework both have practical limitations, which prevent us from applying them to real-world schema-mapping discovery problems. Therefore, we embarked on research that extends, refines, and investigates these two frameworks in more depth. We next discuss the limitations of the two frameworks and summarize our contributions.

1.1.1 Our Contributions

Extending the Gottlob-Senellart framework. In Chapter 2, we present our work on extending and refining the Gottlob-Senellart framework in more depth. We extend the framework by allowing any finite number of ground data examples, instead of a single ground data example. We also refine the framework by considering several different schema-mapping languages. We consider sublanguages $\mathcal{L}$ of the standard language of GLAV constraints. For each of these schema-mapping languages $\mathcal{L}$, we consider two corresponding repair languages, namely, $\mathcal{L}^{=,\neq}$ and $\mathcal{L}^{=}$; the former extends $\mathcal{L}$ with equalities, inequalities, and ground facts (as in [38]), while the latter extends $\mathcal{L}$ with equalities and ground facts only.

The main algorithmic problem in the Gottlob-Senellart framework is to compute an optimal schema mapping for a given ground data example. The tractability of this optimization problem was left open in [38]. We show that this optimization problem is indeed hard for GLAV mappings, for both GLAV$^{=,\neq}$-repairs and GLAV$^{=}$-repairs. More precisely, under a certain assumption, namely, assuming RP $\neq$ NP, there is no polynomial-time algorithm that, given a ground data example, computes a GLAV mapping whose cost is bounded by some fixed polynomial in the cost of the optimal GLAV schema mapping. Moreover, an analogous result hods for GAV mappings. We also designed a $\log(n)$-approximation algorithm for computing near-optimal schema mappings in the more restricted case of GLAV schema mappings. Furthermore, we show that this is a best possible approximation result one could achieve under a certain assumption, namely, assuming P $\neq$ NP. In addition to the complexity-theoretic results, we also implemented the aforementioned $\log(n)$-approximation algorithm and carried out an experimental evaluation in a real-world mapping scenario, where the task is to find a schema mapping from a relational database to an ontology.

A preliminary version of this work appeared in [24], and an extended version of [24] has been accepted by ACM Transaction on Database System in the January of 2017. The co-author listed in [24] directed and supervised the research which forms the basis for this work.

Extending the learning framework. In Chapter 3, we present the a PAC learning algorithm for the class of GAV schema mappings. In fact if an oracle for NP is provided, the proposed algorithm is an efficient PAC learning algorithm for GAV schema mappings. We implemented the proposed learning algorithm and evaluated it using mapping scenarios created by a state-of-the-art benchmarking tool named iBench [8]. As a byproduct of the experimental evaluation, we introduce a new measure (i.e., the F-score measure) that is used for evaluating the quality of schema mappings. By a comparison study of the proposed learning algorithm with two prior methods, we show that our learning algorithm is able to produce better (in terms of F-scores) GAV schema mappings than the other two methods in mapping scenarios synthetically generated by iBench.

1.2 Learning Rule-Based Entity Resolution Algorithms from Data Examples

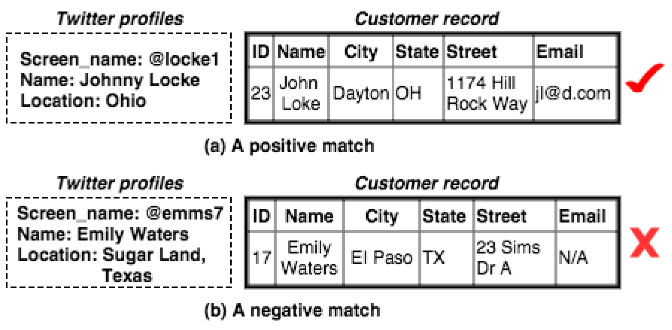
Entity resolution is a hard problem that often requires extensive exploration and understanding of the data domain characteristics even before attempting to formulate the matching algorithms. In many unstructured or semi-structured data scenarios, one needs to know first which attributes are present in the data and how frequently populated, which attributes can actually "identify" an entity and/or are relevant for matching, which are the good comparison functions to apply for the various types of attributes, and so on. As a result, there is a significant amount of human labor that goes in designing and customizing the entity resolution algorithms in a given domain. Furthermore, the scale of the data in modern "big data" applications adds to the complexity of the problem. As an example, assume that the entity resolution task at hand is to identify all matches between a set of Twitter user profiles and a set of Customer (e.g., from Walmart) records. If each set contains about 100 million (10^8) records, then the space of potential matches (the cross product of the two datasets) contains about $10^8 \times 10^8$ records, and becomes prohibitive.

Traditionally, ER has a strong history of applying supervised learning approaches to first learn a model that can be subsequently used to detect duplicates [36]. The availability of fully labeled training data is a pre-condition for applying supervised learning. For ER, this usually implies labeling a large number of examples since the non-matches far outnumber matches and a substantial amount of human effort needs to be invested up front so that the training data consists of more than just a handful of matches. Going back to the previous example of matching Twitter user profiles with customer records, let us further assume that the ratio of matches to non-matches is 1‰ on average. Randomly exploring the space of examples, one would expect to label 10^5 pairs of records in order to obtain a training set consisting of 100 matches. For this reason, recent work in ER has adopted active learning approaches that use a more guided approach to select examples to label [6, 62].

The main idea of active learning in the ER setting is to reduce the number of pairs that need to be labeled by actively selecting the most informative examples. In [62], the user is asked to label pairs of records that lead to maximum disagreement among a committee of statistical classifiers learned by randomizing the input. One shortcoming of these approaches is that they do not provide any quality control over the resulting models. Combined with the necessity of randomization, such approaches lead to very unstable results. In other words, multiple runs over the same input can lead to ER results with drastically varying precision. In contrast, the recent work of Arasu et al [6] uses active learning to learn ER rules whose precision is greater than or equal to a user-defined threshold. The work of Arasu et al [6] has been successful in actively learning a single ER rule with high precision guarantees. Concretely, the focus of [6] is in exploring *likely false positives*, i.e., examples that are covered by the current rule but are likely to be non-matches. By sending such examples to a user for labeling, the learning algorithm can then refine the current rule into a more accurate rule. However, this approach misses some opportunities towards further improving recall; it is also less successful in learning of a second good conjunction or, in general, of a set of conjunctions that are sufficiently different from each other.

1.2.1 Our Contributions

In Chapter 4, we introduce a new active learning system for entity resolution that learns high-quality rule-based entity resolution algorithms at scale. The work presented in Chapter 4 was jointly done with Lucian Popa (IBM Research - Almaden) and Prithviraj Sen (IBM Research - Almaden). In entity resolution, a data example is a pair (r, s) of records where r and s are two records of the input datasets. Moreover, in the setting of ER, every data example can be labeled either as positive (match) or as negative (non-match) by a human user. As an illustration, Figure 1.2 shows two data examples (one positive and one negative) for the Twitter-Customer scenario. As with other active learning in general, the goal of our system is to interleave, in a continuous loop, the generation of ER algorithms with the active exploration of examples, which in turn will lead to the refinement of the rule-based ER algorithms.


Twitter profiles
Customer record
(a) A positive match
Twitter profiles
Customer record
(b) A negative match

Screen_name: @locke1
Name: Johnny Locke
Location: Ohio

ID	Name	City	State	Street	Email
23	John Loke	Dayton	OH	1174 Hill Rock Way	jl@d.com

✓

Screen_name: @emms7
Name: Emily Waters
Location: Sugar Land, Texas

ID	Name	City	State	Street	Email
17	Emily Waters	El Paso	TX	23 Sims Dr A	N/A

✗

Figure 1.2: Examples of matches: (a) positive, (b) negative.

Similar to the work of Arasu et al [6], the ER rules learned by our active learning system have precision greater than or equal to a user-specific threshold. The main difference between our system and the work done by Arasu et al [6] is that, our learning approach actively searches for both likely false positives that are used to improve the precision of current rule, and *likely false negatives*, that is, examples that are not covered by the current rule but are likely positive matches (true matches). The latter type of examples are more challenging to identify since they are outside the space covered by an existing rule; at the same time, they can have a tremendous impact in refining an existing rule towards better recall and in finding additional good rules. One of our main contributions is precisely the ability to generate likely false negatives that in turn, enable learning at scale of multiple ER rules, each having significant coverage (i.e., recall) of the space of true matches.

We summarize our main contributions as follows.

- We propose an active learning system that can learn high quality entity resolution algorithms at scale on big data.
- As part of the overall system, we develop a learning algorithm that can learn a single conjunction (rule) given a set of positive and negative examples. A salient feature of this algorithm is that it maximizes recall while satisfying a high-precision constraint, with respect to the examples.
- We develop techniques to actively search for examples, including both likely false positives and likely false negatives. While determining likely false positives is often

a step in prior active learning algorithms for ER, to the best of our knowledge, we are the first to exploit likely false negatives in a significant way, leading to multiple non-trivial rules.

- We show experimentally that our approach outperforms state-of-the-art methods in terms of quality of the learned algorithms and also in terms of the number of examples that need to be labeled by a human user. The methods that we compare against range from supervised learning methods including an SVM-based pairwise ER classifier and the LSM method that learns Markov Logic Networks (MLNs) [48], to the more closely related active learning method by Arasu et al [6].
- We demonstrate the robustness of our method, by showing that the learned ER rules (i.e., the conjunctions of predicates) produce almost identical sets of matches when evaluated on two very different platforms (both using rules as input): 1) the deterministic HIL-based system [42], running on MapReduce, and 2) a state-of-the art probabilistic inference engine based on Probabilistic Soft Logic (PSL) [11]. Thus, our method provides a general way to learn good quality rules that apply to a variety of entity resolution engines.

Part I

Discovering Schema Mappings

Chapter 2

Discovering Optimal and Near-Optimal Schema Mappings from Data Examples

In this chapter, we contribute to the study of schema-mapping discovery from data examples by refining, extending, and investigating the Gottlob-Senellart framework in more depth. A preliminary version of the work presented in this chapter was published in [24]. We first extend the framework by allowing any finite number of ground data examples, instead of a single ground data example. While, given a finite set E of data examples, there may not be a valid and fully explaining schema mapping for E in the extended language, we give a simple necessary and sufficient condition for the existence of such a schema mapping, a condition that is also easy to verify algorithmically. We also refine the framework by considering several different schema-mapping languages. At the base level, we consider sublanguages $\mathcal{L}$ of the standard language of GLAV constraints, such as the languages of GAV constraints, LAV (Local-As-View) constraints, and SH-

LAV (single-head LAV) constraints. For each of these schema-mapping languages $\mathcal{L}$, we consider two corresponding repair languages, namely, $\mathcal{L}^{=,\neq}$ and $\mathcal{L}^{=}$; the former extends $\mathcal{L}$ with equalities, inequalities, and ground facts (as in [38]), while the latter extends $\mathcal{L}$ with equalities and ground facts only.

The main algorithmic problem in the Gottlob-Senellart framework is to compute an optimal schema mapping for a given ground data example. The tractability of this optimization problem was left open in [38]. We show that this optimization problem is indeed hard for GLAV mappings, for both GLAV$^{=}$-repairs and GLAV$^{=,\neq}$-repairs. More precisely, unless RP = NP, there is no polynomial-time algorithm that, given a ground data example, computes a GLAV mapping whose cost is bounded by some fixed polynomial in the cost of the optimal GLAV mapping. Moreover, an analogous result holds for GAV mappings.

On the positive side, we design a $\log(n)$-approximation algorithm for computing near-optimal schema mappings in the more restricted case of SH-LAV schema mappings. Specifically, we present a polynomial-time $\log(n)$-approximation algorithm that, given a set E of data examples, produces a SH-LAV mapping $\mathcal{M}$ whose cost is within a logarithmic factor of the cost of the optimal SH-LAV mapping for E, together with a witnessing SH-LAV$^{=}$-repair $\mathcal{M}'$ of $\mathcal{M}$ for E. Furthermore, we show that this is a best possible approximation result. SH-LAV schema mappings and the corresponding repair languages form important classes of schema mappings. For instance, many of the primitives of two well-known schema mapping benchmarks can be expressed in these languages (see Section 2.3.2 for a more detailed discussion). In addition to the complexity-theoretic results, we implemented the aforementioned $\log(n)$-approximation algorithm and then tested the implementation on a real-world mapping scenario, where the task is to find a schema mapping from a relational database to an ontology.

The rest of the chapter is organized as follows. In section 2.1, we introduce additional needed concepts and notations (in fact they are both used in Chapter 2 and in Chapter 3). In Section 2.2, we review the repair framework and cost model introduced by Gottlob and Senellart, and also extend it to a finite set of data examples. In Section 2.3, we study the complexity of computing optimal and near-optimal schema mappings. In particular, we first present several negative results on approximating GAV schema mappings (Section 2.3.1) and also show how we can extend the results to the GLAV case (Section 2.3), and then we present the complexity results regarding LAV and SH-LAV schema mappings (Section 2.3.2). In Section 2.4, we present a positive result, that is, a $\log(n)$-approximation algorithm for SH-LAV schema mappings with respect to the repair language SH-LAV$^{=}$; moreover, we show that the approximation factor (i.e., logarithmic factor) cannot be improved (assuming P$\neq$NP). In Section 2.5, we present an experimental evaluation of that approximation algorithm. Finally, in Section 2.6, we summarize our results and discuss open questions and directions for future work.

2.1 Preliminaries

In this section, we introduce the basic notations and concepts needed. Note that the following notations and concepts are used both in Chapter 2 and in Chapter 3.

Schemas and Instances An *instance* over a schema $\mathbf{R} = \{R_1, \ldots, R_k\}$ can be identified with the finite set of all *facts* $R_i(a_1, \ldots, a_m)$, such that R_i is a relation symbol of $\mathbf{R}$ and $(a_1, \ldots, a_m)$ is a tuple that belongs to the relation R_i^I of I interpreting R_i. Database instances contain values that are either *constants* or *nulls*.

A *ground instance* is an instance such all of its facts are *ground*, i.e., they consist entirely of constants. In this chapter, we will primarily consider ground instances, and we will, at times, drop the adjective "ground". We write $\text{adom}(I)$ to denote the *active domain* of an instance I.

A *homomorphism* $h : I_1 \to I_2$ is a function from $\text{adom}(I_1)$ to $\text{adom}(I_2)$ such that for every fact $P(a_1, \ldots, a_m)$ of I_1, we have that $P(h(a_1), \ldots, h(a_m))$ is a fact of I_2.

By convention, a *source instance* refers to an instance of source schema $\mathbf{S}$, and a *target instance* refers to an instance of target schema $\mathbf{T}$.

Schema Mappings Let $\mathbf{S}, \mathbf{T}$ be two relational schemas, called the *source* schema and the *target* schema. A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ consisting of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, and a set Σ of GLAV constraints.

A GLAV (Global-and-Local-As-View) constraint, also known as a *tuple-generating dependency (tgd)*, is a first-order formula of the form

$$\forall \mathbf{x}(\varphi(\mathbf{x}) \to \exists \mathbf{y} \psi(\mathbf{x},\mathbf{y})),$$

where $\varphi(\mathbf{x})$ is a conjunction of atoms over $\mathbf{S}$ and $\psi(\mathbf{x},\mathbf{y})$ is a conjunction of atoms over $\mathbf{T}$. We will often drop the universal quantifiers when writing constraints. There are two important special cases of GLAV constraints:

(1) A GAV (Global-As-View) constraint is a GLAV constraint whose right-hand side is a single atom without existential quantifiers, i.e., it is of the form

$$\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow T(\mathbf{x})).$$

(2) A LAV (Local-As-View) constraint is GLAV constraint whose left-hand side is a single atom, i.e. it is of the form

$$\forall \mathbf{x}(S(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x},\mathbf{y})).$$

There is another special case of LAV constraints, called *SH-LAV (Single-Head LAV) constraints*. A SH-LAV constraint is a GLAV constraint in which both the left-hand side and right-hand side are a single atom, i.e., it is of the form

$$\forall \mathbf{x}(S(\mathbf{x}) \rightarrow \exists \mathbf{y} T(\mathbf{x},\mathbf{y})).$$

Example 2.1 (GLAV, GAV, LAV and SH-LAV constraints) The following is a GLAV constraint:

$$\forall s \forall c\ \ (\text{Student}(s) \wedge \text{Enrolls}(s,c) \rightarrow \exists t \exists g\ \text{Teacher}(t,c) \wedge \text{Grade}(s,c,g))\,.$$

The following constraint is a GAV constraint:

$$\forall u \forall v\ \ (\text{Node}(u) \wedge \text{Node}(v) \rightarrow \text{Edge}(u,v))\,.$$

The following is a LAV constraint that is also a GAV constraint, and hence, in particular, is a SH-LAV constraint:

$$\text{Geo}(x,y) \rightarrow \text{City}(y).$$

The following constraint is a SH-LAV constraint that is not a GAV constraint.

$$\text{Geo}(x, y) \to \exists z \text{City}(z).$$

Finally, the following constraint is a LAV constraint but not a SH-LAV constraint.

$$\forall u \forall v(\text{Edge}(u, v) \to \exists x \ \text{Edge}(u, x) \wedge \text{Edge}(x, v)).$$

$\square$

A *GLAV schema mapping* is a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a finite set of GLAV constraints. The notions of GAV, LAV, and SH-LAV schema mappings are defined in an analogous way.

Data Examples. Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ be a target schema. A *data example* is a pair (I, J) such that I is a source instance that conforms to $\mathbf{S}$ and J is a target instance that conforms to $\mathbf{T}$. A data example (I, J) is *ground* if both I and J are ground instances.

The size $||(I, J)||$ of a data example (I, J) is the total number of facts in I and J. The size $||E||$ of a set of data examples is the sum of the sizes of the data examples in E.

Solutions, Universal Solutions, and Fitting Schema Mappings. We say that a target instance J is a *solution* for a source instance I with respect to (w.r.t.) a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ if $(I, J) \models \Sigma$, that is, (I, J) satisfies every constraint of $\mathcal{M}$.

We say that two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are *logically equivalent*, denoted by $\mathcal{M} \equiv \mathcal{M}'$, if for every pair (I, J) of source instance and target instance, J is a solution for I w.r.t. $\mathcal{M}$ iff J is also a solution for I w.r.t. $\mathcal{M}'$.

We say that a target instance J is a *universal solution* for a source instance I w.r.t. $\mathcal{M}$ if for every other solution J' for I w.r.t. $\mathcal{M}'$, there is a homomorphism from J to J' that is the identity on the active domain of I, i.e., such that $h(a) = a$ for all $a \in adom(I) \cap adom(J)$. In this case, we also say that $\mathcal{M}$ is a *fitting* schema mapping for (I, J). Universal solutions have been considered as the preferred solutions for data exchange [29].

When $\mathcal{M}$ is a GAV schema mapping, then for every source instance I there is a unique target instance J with $adom(J) \subseteq adom(I)$ that is a universal solution for I with respect to $\mathcal{M}$. We call this target instance the *canonical universal solution of I w.r.t. $\mathcal{M}$* and denoted by $\texttt{can-sol}_{\mathcal{M}}(I)$, and it is poly-time computable [29]. The process of computing $\texttt{can-sol}_{\mathcal{M}}(I)$ from I using $\mathcal{M}$ was called as a chase procedure [29].

Note that if J is a canonical universal solution for I w.r.t. a GAV schema mapping $\mathcal{M}$, the pair (I, J) is also called a *universal data example (or, universal example in short)* for $\mathcal{M}$. However, if (I, J) is a universal example for $\mathcal{M}$, J may not to be a canonical universal solution for I w.r.t. $\mathcal{M}$. In chapter 3, we consider universal examples where the target instances are canonical universal solutions for the corresponding source instances. In chapter 3, when we say a data example (I, J) is universal for a GAV schema mapping $\mathcal{M}$, we mean that J is the canonical universal solution for I w.r.t. $\mathcal{M}$.

Canonical Schema Mappings. Let (I, J) be a data example for which there is a fitting GLAV schema mapping. We then define the *canonical GLAV constraint* of (I, J)

is the GLAV constraint

$$\forall \mathbf{x}\big(q_I(\mathbf{x}) \rightarrow \exists \mathbf{y} q_J(\mathbf{x}, \mathbf{y})\big),$$

where $q_I(\mathbf{x})$ is the conjunction of all facts of I (with each value from the active domain of I replaced by a universally quantified variable from $\mathbf{x}$) and $q_J(\mathbf{x}, \mathbf{y})$ is the conjunction of all facts of J (with each value from $\text{adom}(J)\backslash\text{adom}(I)$ replaced by an existentially quantified variable from $\mathbf{y}$). If E is a finite set of data examples over a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$, then the *canonical GLAV schema mapping of* E is the schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ consists of the canonical GLAV constraints of the data examples in E. It follows that the *canonical GAV schema mapping* of E is defined in the same way as the canonical GLAV schema mapping of E, but ignoring all target facts containing nulls. In other words, the canonical GAV schema mapping of E is the canonical GLAV schema mapping of $\{(I, J_\downarrow) \mid (I, J) \in E\}$, where $J_\downarrow$ consists of all facts of J that contain only values from $\text{adom}(I)$. Note that this is indeed a GAV schema mapping.

Validity and Explanation. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping, and let (I, J) be a ground data example. We say that $\mathcal{M}$ is *valid* for (I, J) if $(I, J) \models \Sigma$, that is, (I, J) satisfies all constraints in Σ.

We say that $\mathcal{M}$ *explains* a fact f of J with respect to I if, for all target instances K such that $(I, K) \models \Sigma$, we have that $f \in K$. Finally, we say that $\mathcal{M}$ is *fully explaining* for a data example (I, J) if $\mathcal{M}$ explains each fact of J with respect to I.

In the scenario of multiple data examples, we say that $\mathcal{M}$ is valid for a set E of data examples if for every $(I_k, J_k) \in E$, $(I_k, J_k) \models \Sigma$. We say that $\mathcal{M}$ is fully explaining for E if $\mathcal{M}$ is fully explaining for every data example (I_k, J_k) in E. We will use the expression *vfe* as a shorthand for “valid and fully explaining”.

Example 2.2 Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S} = \{\text{Geo(area, city)}\}$, $\mathbf{T} = \{\text{City(cityName)}\}$, $\Sigma = \{\text{Geo}(x, y) \rightarrow \text{City}(y)\}$. Consider the data examples (I_i, J_i), $i = 1, 2, 3$, where

$$\begin{aligned}
&I_1 : \{\text{Geo(CA, San Francisco)}, \text{Geo(CA,San Jose)}, \text{Geo(US,Boston)}, \text{Geo(US, Los Angeles)}\}\\
&J_1 : \{\text{City(San Francisco)}, \text{City(San Jose)}\}\\
&I_2 : \{\text{Geo(CA, San Francisco)}\}\\
&J_2 : \{\text{City(San Francisco)}, \text{City(New York)}\}\\
&I_3 : \{\text{Geo(CA, San Francisco)}, \text{Geo(US,New York)}\}\\
&J_3 : \{\text{City(San Francisco)}, \text{City(New York)}\}
\end{aligned}$$

In these data examples, since I_1 contains Geo(US,Boston), but City(Boston) is not in J_1, we have that $\mathcal{M}$ is not valid for (I_1, J_1); moreover, $\mathcal{M}$ is valid for (I_2, J_2), but not fully explaining, as it fails to explain the target fact City(New York); however, $\mathcal{M}$ is valid and fully explaining for (I_3, J_3). Using a simple automorphism argument, it can be shown that there is no valid and fully explaining GLAV schema mapping for (I_1, J_1). Similarly, since J_2 contains a constant that does not appear in the I_2, there is no valid and fully explaining GLAV schema mapping for (I_2, J_2).

□

2.2 Repair Framework and Cost Model

In [38], the language of GLAV constraints was used as the “base language”, and an extended “repair language” was introduced; the repair language includes equalities, inequalities, and ground facts, and is used for “repairing” GLAV schema mappings, so that they become valid and fully explaining for a given ground data example. We refine this framework by considering different sublanguages of the language of GLAV constraints, as well different repair languages.

Definition 2.3 Fix a schema-mapping language $\mathcal{L}$ (e.g., GLAV). An $\mathcal{L}^{=,\neq}$*-constraint* is a formula θ of the form $\forall\mathbf{x}(\varphi(\mathbf{x}) \wedge \alpha(\mathbf{x}) \to \exists\mathbf{y}(\psi(\mathbf{x},\mathbf{y}) \wedge \beta(\mathbf{x},\mathbf{y})))$, where

1. $\forall\mathbf{x}(\varphi(\mathbf{x}) \to \exists\mathbf{y}\psi(\mathbf{x},\mathbf{y}))$ is an $\mathcal{L}$-constraint;
2. $\alpha(\mathbf{x})$ is a possibly empty conjunction of formulas of the form $x \sim c$, where

   $\sim\ \in \{=, \neq\}$, $x \in \mathbf{x}$, and c is a constant;
3. $\beta(\mathbf{x},\mathbf{y})$ is a possibly empty conjunction of formulas of the form

   $$(x_1 = c_1 \wedge \ldots \wedge x_n = c_n) \to y = c,$$

   where each $x_i \in \mathbf{x}$, $y \in \mathbf{y}$, and $c_1, \ldots, c_n, c$ are constants.

An $\mathcal{L}^{=}$*-constraint* is an $\mathcal{L}^{=,\neq}$-constraint with no conjuncts of the form $x \neq c$ in α.

In what follows, we will continue using $\mathcal{L}$ to denote a schema-mapping language (e.g., GLAV), and we will write $\mathcal{L}^{=,\neq}$ and $\mathcal{L}^{=}$ to denote the corresponding repair language (with and without inequalities). We will use $\mathcal{L}^*$ to refer to either $\mathcal{L}^{=,\neq}$ or $\mathcal{L}^{=}$. We say that the formula θ as above is a $\mathcal{L}^*$*-repair* of the constraint $\forall\mathbf{x}(\varphi(\mathbf{x}) \to \exists\mathbf{y}\psi(\mathbf{x},\mathbf{y}))$.

The definition of $\mathcal{L}^{=,\neq}$ above was given in [38], and extensive justification for this repair language was given in that work. The significance of $\mathcal{L}^{=}$ (introduced here) will become clear later on.

Definition 2.4 An $\mathcal{L}^*$*-repair* of an $\mathcal{L}$-schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is an $\mathcal{L}^*$-schema mapping $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ such that each $\psi \in \Sigma'$ is either a ground target fact or is an $\mathcal{L}^*$-repair of some $\phi \in \Sigma$, and, for each $\phi \in \Sigma$, at least one $\mathcal{L}^*$-repair of ϕ belongs to Σ'. We write $\text{repair}_{\mathcal{L}^*}(\mathcal{M})$ to denote the set of all $\mathcal{L}^*$-repairs of $\mathcal{M}$.

The above definition differs from that of repairs in [38] in that we allow $\mathcal{M}'$ to contain multiple repairs of the same $\mathcal{L}$-constraint from $\mathcal{M}$, whereas, in [38], the $\mathcal{L}^*$-constraints of $\mathcal{M}'$ stand in a one-to-one correspondence with the $\mathcal{L}$-constraints of $\mathcal{M}$. There are cases in which an $\mathcal{L}$-constraint may need to be repaired more than once with, say, different combinations of equalities and inequalities. In such cases, the optimal schema mapping in [38] may contain multiple copies of the same GLAV constraint or multiple GLAV constraints that are identical up to a renaming of variables (see Example 2.6). Our definition addresses this shortcoming. It does not impact our complexity results.

Example 2.5 Recall that the data example (I_1, J_1) in Example 2.2 had no vfe GLAV schema mapping. Let $\mathbf{S} = \{\text{Geo(area,city)}\}$ and $\mathbf{T} = \{\text{City(cityName)}\}$. Consider the GLAV$^{=,\neq}$ schema mappings $\mathcal{M}_k = (\mathbf{S}, \mathbf{T}, \Sigma_k)$, where Σ_k $(k = 1, \ldots, 5)$ is

- Σ_1=$\{\text{Geo}(x, y) \wedge (x = \text{CA}) \rightarrow \text{City}(y)\}$;
- Σ_2=$\{\text{Geo}(x, y) \rightarrow \exists z\ \text{City}(z) \wedge (y = \text{San Jose} \rightarrow z = \text{San Jose}) \wedge (y = \text{San Francisco} \rightarrow z = \text{San Francisco})\}$;
- Σ_3=$\{\text{City(San Francisco)}, \text{City(San Jose)}\}$;
- Σ_4=$\{\text{Geo}(x, y) \wedge (x \neq \text{US}) \rightarrow \text{City}(y)\}$;
- Σ_5=$\{\text{Geo}(x, y) \wedge (x = \text{moon}) \rightarrow \text{City}(x)\}$

The mappings $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, and $\mathcal{M}_4$ are vfe for (I_1, J_1), but are obtained in different ways: $\mathcal{M}_1$ consists of a repair of $\text{Geo}(x, y) \rightarrow \text{City}(y)$ that uses an equality to restrict the constraint to source facts whose first attribute has value CA; $\mathcal{M}_2$ consists of a repair of

Table 2.1: Data Example (I, J)

Source Instance		**Target Instance**
Geo(US, San Francisco)	Geo(Calif, San Jose)	City(San Francisco)
Geo(US, Los Angeles)	Geo(Calif, San Francisco)	City(San Jose)
Geo(US, Miami)	Geo(Calif, Los Angeles)	City(San Diego)
Geo(US, Boston)	Geo(Calif, San Diego)	City(Los Angeles)
Geo(US, New York)	Geo(Canada, Vancouver)	City(Boston)
Geo(NorthAm, Boston)	Geo(Germany, Berlin)	City(Toronto)
Geo(NorthAm, Toronto)	Geo(Japan, Tokyo)	City(New York)
Geo(NorthAm, New York)	Geo(China, Beijing)	City(Miami)
Geo(NorthAm, Miami)	Geo(France, Paris)	
	Geo(UK, London)	

$\text{Geo}(x, y) \rightarrow \exists z \text{City}(z)$ that uses two conditional equalities to explicitly specify a value of z depending on the value of y; $\mathcal{M}_3$ is a repair of the empty schema mapping, and it lists all the ground facts of J_1; finally, $\mathcal{M}_4$ consists of a repair of the same constraint as $\mathcal{M}_1$ that, instead, uses an inequality. $\mathcal{M}_5$ is valid, but not fully explaining for (I_1, J_1). It uses the equality $(x = \textit{moon})$, where *moon* is outside the active domain of (I_1, J_1). We informally say that this equality *cancels* $\mathcal{M}_5$ (with respect to the data example (I_1, J_1)). $\square$

Example 2.6 The following examples illustrates why, in Definition 2.4, we allow for multiple repairs of the same constraint. Consider the data example (I, J) in Table 2.1, and consider the GLAV schema mappings

- $\mathcal{M}_1 = \{\text{Geo}(x, y) \rightarrow \text{City}(y)\}$,
- $\mathcal{M}_2 = \{\text{Geo}(x_1, y_1) \rightarrow \text{City}(y_1), \text{Geo}(x_2, y_2) \rightarrow \text{City}(y_2)\}$.

Consider the GLAV$^{=,\neq}$-schema mapping $\mathcal{M}_r$ specified by the constraints

$$\text{Geo}(x, y) \wedge (x = \text{Calif}) \rightarrow \text{City}(y) \quad \text{and} \quad \text{Geo}(x, y) \wedge (x = \text{NorthAm}) \rightarrow \text{City}(y).$$

Note that $\mathcal{M}_r$ is vfe for (I, J). Note also that $\mathcal{M}_2$ consists of two renamings of the GLAV constraint of $\mathcal{M}_1$. By our definition of repair, $\mathcal{M}_r$ is a repair of $\mathcal{M}_1$ and also of $\mathcal{M}_2$. However, according to the definition of repair in [38], $\mathcal{M}_r$ does not qualify as a repair of $\mathcal{M}_1$, and, indeed, an example of a minimal-size repair of $\mathcal{M}_1$ is obtained by adding an equality (x = US) to the left-hand side of the constraint in $\mathcal{M}_1$ together with adding three ground facts: City(San Jose), City(Toronto), and City(San Diego). Since the cost of a schema mapping w.r.t. a data example will be measured by the size of the smallest repair (see Definitions 2.9 and 2.10 below), we have that, under the cost model of [38], $\mathcal{M}_2$ has a lower cost than $\mathcal{M}_1$, which is counterintuitive. The definition of cost we use here solves this problem. $\square$

As pointed out in [38], if (I, J) is a ground data example, then there is always a vfe GLAV$^{=,\neq}$ schema mapping for (I, J). In fact, trivially, the collection of all the ground facts in J is a vfe schema mapping for (I, J). This shows that, for all schema-mapping languages $\mathcal{L}$ considered here, every ground data example has a vfe $\mathcal{L}^{=}$ repair. Indeed, for every data example (I, J) it holds that every $\mathcal{L}$-schema mapping has a $\mathcal{L}^{=}$-repair that is vfe (which can be obtained, for instance, by cancelling all constraints and adding all ground target facts). However, the same result does not hold for a set of ground data examples (note that the trivial schema mapping consisting of all the ground target facts occurring in the input set of data examples is not always vfe). Theorem 2.7, below, establishes a necessary and sufficient condition for the existence of a vfe GLAV$^{=,\neq}$ (resp. GLAV$^{=}$) schema mapping for a set E of data examples.

Theorem 2.7 Let E be a finite set of ground data examples. There exists a vfe

GLAV$^{=,\neq}$ schema mapping for E if and only if the following condition holds: for every pair of data examples $(I,J),(I',J') \in E$, if $I \subseteq I'$, then $J \subseteq J'$. The same holds true when GLAV$^{=,\neq}$ is replaced by GLAV$^{=}$. Moreover, this condition is checkable in polynomial time.

Note that, in particular, Theorem 2.7 implies that GLAV$^{=}$ and GLAV$^{=,\neq}$ are, in a sense, equally powerful repair languages: a finite set of ground data examples has a vfe GLAV$^{=}$ schema mapping if and only if it has a vfe GLAV$^{=,\neq}$ schema mapping. Before we give the proof of Theorem 2.7, we need a definition:

Definition 2.8 The complete-description constraint of a ground data example (I,J) is the GLAV$^{=}$ constraint t that is of the form $\forall \mathbf{x}(q_I(\mathbf{x}) \wedge \alpha(\mathbf{x}) \rightarrow \exists \mathbf{y} q_J(\mathbf{y}) \wedge \beta(\mathbf{y}))$, where $q_I(\mathbf{x})$ is the conjunction of all facts of I (with each value from the active domain of I replaced by a universally quantified variable from $\mathbf{x}$) and $q_J(\mathbf{y})$ is the conjunction of all facts of J (with each value replaced by an existentially quantified variable from $\mathbf{y}$); $\alpha(\mathbf{x})$ is the conjunction of all equalities $(x_i = c_i)$, where x_i is the variable corresponding to c_i, and likewise for $\beta(\mathbf{y})$. For instance, if $I = \{S(a), S(b)\}$ and $J = \{T(a,b)\}$, then the complete-description constraint of (I,J) is $\forall xy(S(x) \wedge S(y) \wedge (x = a) \wedge (y = b) \rightarrow \exists wu(T(w,u) \wedge (w = a) \wedge (u = b))$.

Proof of Theorem 2.7. ($\Leftarrow$) Suppose the condition holds, and consider the set Σ consisting of the complete-description constraint of each data example $(I_k, J_k) \in E$. We show that the schema mapping specified by Σ is vfe for E.

Validity. We need to show that for each data example $(I_k, J_k) \in E$ and for each

$\sigma \in \Sigma$, we have $(I_k, J_k) \models \sigma$. By construction, σ is the complete-description constraint of some data example $(I_\sigma, J_\sigma) \in E$. If $I_k \not\subseteq I_\sigma$, then σ is trivially valid in (I_k, J_k), because facts of I_k would never fire the left-hand side of σ. If, on the other hand, $I_\sigma \subseteq I_k$, then, by assumption $J_\sigma \subseteq J_k$, and hence σ is valid in (I_k, J_k)., We conclude that Σ is valid for E.

Explanation. We need to show that Σ fully explains every data example of E. Consider any $(I, J) \in E$, and let σ be the complete-description constraint of of (I, J). In order to show that σ fully explains (I, J), we need to show that for every instance K such that $(I, K) \models \sigma$, we have $J \subseteq K$. Since $(I, K) \models \sigma$,we have that the homomorphism from the left-hand side of σ to I can be extended to a homomorphism from the right-hand side of σ to K. By construction of σ, it follows that $J \subseteq K$, and thus σ fully explains (I, J). Since the same argument applies to each data example in E, we have that the schema mapping specified by Σ fully explains E.

$(\Rightarrow)$. Suppose that there is a vfe schema mapping $\mathcal{M}$ for E, and consider any pair of data examples (I, J), (I', J') in E, such that $I \subseteq I'$. We have that, in this case, J' must be also a solution for I w.r.t. $\mathcal{M}$ (because every homomorphism from the left-hand side of a constraint to I is also a homomorphism from the left-hand side of the same constraint to I'). Since $\mathcal{M}$ is fully explaining for E, we conclude that $J \subseteq J'$. Since no inequality is used, the same holds for when GLAV$^{=,\neq}$ is replaced by GLAV$^{=}$.

The condition is checkable in polynomial time, because if there are n data examples in E where each data example has at most n facts, the checking function needs to compare n^2 pairs of examples. In each comparison, we need to examine at most $4n$ facts, thus this can be done in polynomial time. □

The cost model introduced in [38] focuses on finding a schema mapping in the base language, such that the cost of transforming it to a vfe schema mapping in the repair language is as small as possible.

Definition 2.9 [38] The *size* of a first-order formula φ, denoted size(φ), is the number of occurrences of variables and constants in φ (each variable and constant is counted as many times as it occurs in φ); occurrences of variables as arguments of quantifiers do not count. A ground fact $R(a_1, \ldots, a_n)$, for present purposes, is viewed as shorthand for $\exists x_1, \ldots, x_n R(x_1, \ldots, x_n) \wedge x_1 = a_1 \wedge \cdots \wedge x_n = a_n$; therefore, we consider its *size* to be $3n$. The *size* of a repair of a schema mapping is the sum of the sizes of the constraints and the ground facts of the repair.

Note that the motivation for the above treatment of ground facts is to discourage the use of ground facts in repairing schema mappings.

Definition 2.10 The *cost* of an $\mathcal{L}$-schema mapping $\mathcal{M}$ for a set E of data examples and a repair language $\mathcal{L}^*$, denoted by cost($\mathcal{M}, E, \mathcal{L}^*$), is the smallest size of a vfe $\mathcal{L}^*$-repair of $\mathcal{M}$ for E. An $\mathcal{L}$-schema mapping $\mathcal{M}$ is *$\mathcal{L}^*$-optimal for E* if cost($\mathcal{M}, E, \mathcal{L}^*$) is the minimum of the quantities cost($\mathcal{M}', E, \mathcal{L}^*$), where $\mathcal{M}'$ ranges over all $\mathcal{L}$-schema mappings.

Example 2.11 We continue from Example 2.5 by considering the schema mappings $\mathcal{M}_a = \{\text{Geo}(x, y) \rightarrow \text{City}(y)\}$, $\mathcal{M}_b = \{\text{Geo}(x, y) \rightarrow \exists z \ \text{City}(z)\}$, and $\mathcal{M}_c = \emptyset$.

Both $\mathcal{M}_1$ and $\mathcal{M}_4$ are vfe repairs of $\mathcal{M}_a$ for (I_1, J_1) and both have size 5; in fact no other vfe repairs of $\mathcal{M}_a$ has size less than 5, hence $\text{cost}(\mathcal{M}_a, \{(I_1, J_1)\}, \text{GLAV}^{=,\neq}) = 5$. The repair $\mathcal{M}_3$ is the only vfe repair of $\mathcal{M}_c$, and $\text{cost}(\mathcal{M}_c, \{(I_1, J_1)\}, \text{GLAV}^{=,\neq}) = 6$ (recall that the size of a ground fact is three times its arity). Moreover, $\mathcal{M}_2$ is a vfe repair of $\mathcal{M}_b$ and $\text{size}(\mathcal{M}_2) = 11$, but the cost of $\mathcal{M}_b$ is not 11. To see this, consider the schema mapping $\mathcal{M}_2'$ specified by the constraint

$$\mathcal{M}_2' = \{\text{Geo}(x, y) \rightarrow \exists z \ \text{City}(z) \wedge (z = \text{San Jose}), \text{City}(\text{San Francisco})\}.$$

Clearly, $\mathcal{M}_2'$ is also a vfe repair of $\mathcal{M}_b$ for (I_1, J_1) and has size of 8. □

Recall that we consider a slightly different notion of repair than the one in [38], as explained in the remarks following Definition 2.4. As a consequence, the cost of a schema mapping, under our cost model, may be less than the cost under the cost model in [38]. Nevertheless, every optimal schema mapping in our cost model has the same cost as the cost of an optimal schema mapping in the cost model of [38]. However, the complexity-theoretic results concerning the various algorithmic problems do not depend on this, and, indeed, the proofs are not affected by this change.

2.3 Hardness of Computing Optimal and Near-Optimal Schema Mappings

As we have seen, the main idea of the framework introduced in [38] and refined here, is to cast schema-mapping discovery as an optimization problem: given a finite set of ground data examples, produce a schema mapping of minimum cost. Two different decision problems naturally arise in this setting.

Definition 2.12 The decision problem $\textsc{Cost}_{\mathcal{L}^*}$ asks: given a ground data example (I, J) given a set E of ground data examples, a $\mathcal{L}$-schema mapping $\mathcal{M}$, and an integer $k \geq 0$, is $\text{cost}(\mathcal{M}, E, \mathcal{L}^*) \leq k$?

Definition 2.13 The decision problem $\textsc{Existence-Cost}_{\mathcal{L}^*}$ asks: given a ground data example (I, J) given a set E of ground data examples and an integer $k \geq 0$, does there exist a $\mathcal{L}$-schema mapping $\mathcal{M}$ such that $\text{cost}(\mathcal{M}, E, \mathcal{L}^*) \leq k$

In these two problems, the source schema and the target schema are part of the input. One can also consider the variants of these problems, such as $\textsc{Existence-Cost}_{\mathcal{L}^*}(\mathbf{S}, \mathbf{T})$, obtained by fixing the source and target schemas.

The above problems (where E consists of a single ground data example) were introduced and studied in [38] for the case where the base language $\mathcal{L}$ is GLAV or GAV, and the repair language is $\mathcal{L}^{=,\neq}$. It was shown there that $\textsc{Cost}_{\text{GLAV}^{=,\neq}}$ belongs to Σ_3^p (the third level of the polynomial hierarchy) and is Π_2^p-hard, while $\textsc{Cost}_{\text{GAV}^{=,\neq}}$ belongs to Σ_2^p and is DP-hard. It was also shown that $\textsc{Existence-Cost}_{\text{GLAV}^{=,\neq}}$ belongs to Σ_3^p and that $\textsc{Existence-Cost}_{\text{GAV}^{=,\neq}}$ belongs to Σ_2^p; moreover, both these problems were shown to be NP-hard[1] These results are also summarized in Table 2.2.

The $\textsc{Cost}$ and $\textsc{Existence-Cost}$ problems for LAV schema mappings and its sub-language SH-LAV schema mappings were not considered in [38]. In addition, exploring the approximation properties of different schema mappings was left as a future work in [38]. In the rest of this paper, we study the aforementioned open questions in

[1]The NP-hardness proof given in [38] is flawed. The authors have shared with us, in private communication, a correct NP-hardness proof.

Table 2.2: Relevant complexity results in [38]

Decision problem	**complexity results**
$\textsc{Cost}_{\mathrm{GLAV}^{=,\neq}}$	in Σ_3^p, Π_2^p-hard
$\textsc{Cost}_{\mathrm{GAV}^{=,\neq}}$	in Σ_2^p, DP-hard
$\textsc{Existence-Cost}_{\mathrm{GLAV}^{=,\neq}}$	in Σ_3^p, NP-hard
$\textsc{Existence-Cost}_{\mathrm{GAV}^{=,\neq}}$	in Σ_2^p, NP-hard

more depth. The main results, in this section, are:

1. We show that (assuming RP $\neq$ NP) there is no polynomial time algorithm that, given a ground data example, computes an optimal GLAV schema mapping, or even a GLAV schema mapping whose cost is bounded by a fixed polynomial in the cost of the optimal GLAV schema mapping. The same result holds for GAV schema mappings, regardless of the repair language considered (see Theorem 2.14).

2. We show that the Cost and Existence-Cost problems are NP-complete for LAV schema mappings and SH-LAV schema mappings, regardless of the repair language considered (see Theorem 2.23).

2.3.1 Hardness of Computing Optimal and Near-Optimal GAV Schema Mappings

We first show that the problem of computing optimal GLAV schema mappings is not solvable in polynomial time, unless RP = NP. Note that this does not follow directly from the NP-hardness of $\textsc{Existence-Cost}_{\mathrm{GLAV}^{=,\neq}}$ established in [38], because $\textsc{Cost}_{\mathrm{GLAV}^{=,\neq}}$ is Π_2^p-hard.

Actually, we show a stronger result: unless RP = NP, there is no polynomial-time algorithm that, given a ground data example (I, J), computes a GLAV schema mapping whose cost is bounded by a polynomial in the cost of the optimal GLAV schema mapping; moreover, the same holds true for GAV schema mappings.

Theorem 2.14 Let $\mathcal{L} \in \{GLAV, GAV\}$, let $\mathcal{L}^* \in \{\mathcal{L}^=, \mathcal{L}^{=,\neq}\}$, and let $p(x)$ be any polynomial. Unless RP = NP, there is no polynomial-time algorithm that, given a ground data example (I, J), computes a $\mathcal{L}$-schema mapping $\mathcal{M}$ such that $\text{cost}(\mathcal{M}, (I, J), \mathcal{L}^*) \leq p(\text{cost}(\mathcal{M}_{opt}, (I, J), \mathcal{L}^*))$, where $\mathcal{M}_{opt}$ is an $\mathcal{L}^*$-optimal schema mapping for (I, J). In fact, there are fixed source and target schemas for which this result holds.

Theorem 2.14 also shows that the computational hardness is, to some extent, robust under changes to the precise cost function.

The proof of Theorem 2.14 will make use of a result from [21] (the preliminary version appeared in [20]); we spell that result first and then give a roadmap for the proof of the desired Theorem 2.14.

We say that a class of data examples is *GLAV-tractable* if there is a polynomial time algorithm that, given as input a data example from the class and a GLAV-constraint, checks if the data example satisfies the GLAV-constraint.

Theorem 2.15 (from [21]) There exist schemas $\mathbf{S}$ and $\mathbf{T}$ and a GLAV-tractable class K of data examples over $\mathbf{S}$ and $\mathbf{T}$, where $\mathbf{T}$ consists of unary relations, such that the following holds: let $p(x)$ be any polynomial function. Unless RP = NP, there is no polynomial-time algorithm that, given a ground data example $(I, J) \in K$ for which there exists vfe GAV-schema mappings, computes a vfe GAV-schema mapping $\mathcal{M}$ such that

$size(\mathcal{M}) \leq p(size(\mathcal{M}_{\min}))$, where $\mathcal{M}_{\min}$ is the minimal-size vfe GAV-schema mapping for (I, J).

Proof. The result was proved in [21] without reference to GLAV-tractability. Careful inspection of the proof in [21] shows that the class of source instances used is CQ-tractable (defined below), and that the target schema is unary. As we will show, this implies the data examples are GLAV-tractable. We say that a class C of instances (for a schema $\mathbf{S}$) is *CQ-tractable* if there is a polynomial-time algorithm that given an instance $I \in C$, an n-ary conjunctive query q ($n \geq 0$), and a tuple $\mathbf{a} \in adom(I)^n$, decides if $\mathbf{a} \in q(I)$. Note that CQ-tractable classes of instances are rare. Indeed, the classic reduction from 3SAT to conjunctive query evaluation shows that there is singleton class C consisting of a two-element instance, such that C is CQ-intractable unless P = NP. A non-trivial example of a CQ-tractable class of instances is the class of disjoint unions of oriented paths (viewed as instances over a schema consisting of a single binary relation) [40].

Let K be a class of data examples over source and target schema $\mathbf{S}$ and $\mathbf{T}$, such that the source instances of the data examples in K belong to a CQ-tractable class, and $\mathbf{T}$ contains only unary relations. We will show that K is a GLAV-tractable class of data examples.

Let (I, J) be a data example from K and consider any GLAV-constraint t of the form $\forall \mathbf{x}\mathbf{y}\ \varphi(\mathbf{x}, \mathbf{y}) \rightarrow\ \exists \mathbf{z}\ \psi(\mathbf{x}, \mathbf{z})$. We can split t into a finite set Σ of GLAV-constraints such that each GLAV-constraint in Σ has only one variable occurring in the right-hand side. Concretely, for each GLAV-constraint t with m variables occurring in the right-hand side, where $m > 1$, we split t into m GLAV-constraints in the following way: for each variable z occurring in the right-hand side of t, we create a new GLAV-constraint t' with identical left-hand side as t, and the right-hand side of t' contains only those atoms that use the variable z. After this splitting, each GLAV-constraint is of the form (1) $\forall \mathbf{x} y\ \varphi(\mathbf{x}, y) \to\ \psi(y)$, or (2) $\forall \mathbf{x}\ \varphi(\mathbf{x}) \to\ \exists y\ \psi(y)$, where ψ is a conjunction of relational atoms. Note that t and Σ are logically equivalent, and the combined size of the constraints in Σ is $O(n^2)$ where n is the size of t. It therefore suffices to show that we can check the validity of the constraints from Σ in (I, J) in polynomial time. This follows immediately from the fact that I belongs to a CQ-tractable class: we can evaluate the left-hand side $\exists \mathbf{x} \varphi$ in I in polynomial time, and check that the right-hand side is satisfied. $\square$

We now provide a roadmap of the proof of Theorem 2.14. The proof proceeds by contradiction: we show that if there is a polynomial-time algorithm A that solves the problem described in Theorem 2.14, then, using A as an oracle, we can also solve the problem described in Theorem 2.15 in polynomial time. More precisely, our construction is based on a procedure that transforms a ground data example (I, J) (for which there is a vfe GAV schema mapping) into another ground data example (I', J') that contains many isomorphic copies of the original data example (I, J). Intuitively, after creating "sufficiently" many isomorphic copies of (I, J) in (I', J'), every GAV schema mapping $\mathcal{M}$ that needs to be repaired so as to be vfe for (I, J) will have to contain a large number of repairs so as to be vfe for (I', J') (see Lemma 2.20). As a result, if the hypothetical

Algorithm A returns a near-optimal GAV schema mapping $\mathcal{M}'$ for (I', J'), then a "small" vfe GAV schema mapping $\mathcal{M}_{\text{small}}$ for (I, J) can be extracted from $\mathcal{M}'$ in polynomial time (see Lemma 2.20 and Proposition 2.21). We show that $\mathcal{M}_{\text{small}}$ is a solution for the problem in Theorem 2.15, and the contradiction is established.

The rest of the section is dedicated to a detailed proof of Theorem 2.14. We will give a detailed proof for the GAV$^{=,\neq}$ case, which applies also to the GAV$^{=}$ case. In Section 2.3, we will explain how the proof can be modified to be a proof for the GLAV$^{=,\neq}$ and GLAV$^{=}$ case.

Lemma 2.16 ([3]) Let E be any finite set of data examples. If there is a vfe GAV schema mapping for E, then there is one of size at most $||E||^2$.

We say that a GAV constraint is *connected* if the graph, where the nodes are the atoms (both on the left-hand side and on the right-hand side) in the constraint and two atoms are connected by an edge if they have a variable in common, is connected. Thus, for example, the GAV constraints $\forall xy(R(x, y) \rightarrow S(x, y))$ and $\forall xyP(x) \wedge Q(y) \rightarrow R(x, y)$ are connected, while the GAV constraint $\forall xyzv(R(x, y) \wedge U(z, v) \rightarrow S(x, y))$ is not connected. We say that a GAV schema mapping is connected if it consists of connected GAV constraints.

Lemma 2.17 Every GAV schema mapping $\mathcal{M}$ that is a minimal-size vfe GAV schema mapping for a data example (I, J) is connected.

Proof. Let $\mathcal{M}$ be a minimal vfe GAV schema mapping for (I, J), and suppose, for the sake of a contradiction, that $\mathcal{M}$ is not connected. If a GAV constraint is not connected, (the graph of) its atoms can be divided into two or more connected components. One of these connected components must contain the right-hand side of the constraint. Let $\widehat{\mathcal{M}}$ be obtained from $\mathcal{M}$ simply by removing, from left-hand side of each GAV constraint, all atoms that do not belong to the connected component containing the right-hand side. Note that $\widehat{M}$ is strictly smaller than $\mathcal{M}$. We will show that $\widehat{\mathcal{M}}$ is vfe for (I, J), which contradicts the assumption that $\mathcal{M}$ was a minimal vfe GAV schema mapping for (I, J).

Since $\mathcal{M}$ is assumed to be a minimal vfe schema mapping for (I, J), every constraint σ of $\mathcal{M}$ explains at least one target fact. This implies that there is at least one homomorphism h from the left-hand side of σ to I. Let $\widehat{\sigma}$ be the connected constraint of $\widehat{\mathcal{M}}$ that was obtained from σ. Clearly, every target fact that is explained by σ is also explained by $\widehat{\sigma}$. Conversely, every target fact explained by $\widehat{\sigma}$ is explained by σ, because every homomorphism g from the left-hand side of $\widehat{\sigma}$ to I can be extended to a homomorphism g' from the left-hand side of σ to I (where g' simply extends g by mapping every variable $x \in dom(h) \setminus dom(g)$ to $h(x)$). It follows that $\mathcal{M}$ and $\widehat{\mathcal{M}}$ explain the same target facts, and hence $\widehat{\mathcal{M}}$ is vfe for (I, J). $\square$

Lemma 2.18 For each GAV schema mapping $\mathcal{M}$, there is a connected GAV schema mapping $\mathcal{M}'$, such that, for all data examples (I, J),

$$\text{cost}(\mathcal{M}', \{(I, J)\}, GAV^{=,\neq}) \leq 2 \cdot \text{cost}(\mathcal{M}, \{(I, J)\}, GAV^{=,\neq})$$

and the same holds for when $GAV^{=,\neq}$ is replaced by $GAV^{=}$. Moreover, in both cases,

$\mathcal{M}'$ can be computed from $\mathcal{M}$ in polynomial time.

Proof. If a GAV constraint is not connected, (the graph of) its atoms can be divided into two or more connected components. One of these connected components must contain the right-hand side of the constraint. Let $\widehat{\mathcal{M}}$ be obtained from $\mathcal{M}$ simply by removing, from left-hand side of each GAV constraint, all atoms that do not belong to the connected component containing the right-hand side. This can clearly be done in polynomial time. We claim that $\widehat{\mathcal{M}}$ satisfies the requires properties.

Let (I, J) be an arbitrary data example and $\mathcal{M}_r$ be a minimal vfe repair of $\mathcal{M}$ with respect to (I, J). We have to show that there is a repair $\widehat{\mathcal{M}_r}$ of $\widehat{\mathcal{M}}$ such that $\widehat{\mathcal{M}_r}$ is vfe with respect to (I, J), and such that the size of $\widehat{\mathcal{M}_r}$ is at most twice the size of $\mathcal{M}_r$. We take

$$\widehat{\mathcal{M}_r} = \{\widehat{\psi} \mid \psi \in \mathcal{M}_r\}$$

where, for each repaired GAV constraint $\psi \in \mathcal{M}_r$, $\widehat{\psi}$ as follows: let $\phi \in \mathcal{M}$ be the GAV constraint of which ψ is a repair, and let $\widehat{\phi}$ be the corresponding GAV constraint in $\widehat{\mathcal{M}}$ (that is, $\widehat{\phi}$ is obtained from ϕ by removing disconnected atoms from the left-hand side). We distinguish two cases:

First, we consider the case where ψ explains at least one fact of J. In particular, this means that there is a homomorphism h from the left-hand side of ψ into I. In this case, we simply let $\widehat{\psi}$ be the repair of $\widehat{\phi}$ obtained by applying the same repair operations as in ψ, insofar as they concern variables that still belong to $\widehat{\phi}$. Clearly, all atoms in the left-hand side of $\widehat{\psi}$ occur also in the left-hand side of ψ. Therefore, $\widehat{\psi}$ explains all facts of J explained by ψ. Moreover, $\widehat{\psi}$ is valid in (I, J) because every homomorphism g from the left-hand side of $\widehat{\psi}$ can be extended to a homomorphism g' from the left-hand side of ψ to I (where g' simply extends g by mapping every variable $x \in dom(h) \setminus dom(g)$ to $h(x)$).

Next, consider the case where ψ does not explain any fact of J. Since $\mathcal{M}_r$ is valid with respect to (I, J), this implies that there is no homomorphism from the left-hand side of ψ to I (because, if there were, then the right-hand side of ψ would be satisfied in J, and hence ψ would explain at least one fact of J). If ϕ is connected (and hence $\widehat{\phi} = \phi$), we simply take $\widehat{\psi} = \psi$. Otherwise, we construct $\widehat{\psi}$ from $\widehat{\phi}$ by adding a single equality with a constant outside the active domain of I. We then have that the size of $\widehat{\psi}$ is at most twice the size of ψ, and ψ is trivially valid in (I, J).

In each of the above cases, $\widehat{\psi}$ was constructed so that it is valid with respect to (I, J) and it explains all facts of J explained by ψ, while at the same time having a size at most twice the size of ψ. It follows that the cost of $\widehat{\mathcal{M}}$ with respect to (I, J) is at most twice as large as the cost of $\mathcal{M}$ with respect to (I, J). $\square$

We will prove Theorem 2.14 by contradiction: if a poly-time algorithm A for the problem in Theorem 2.14 exists, then, using A as an oracle, we can solve the problem in Theorem 2.15 in polynomial-time as well.

Let p be a fixed polynomial. Take an arbitrary ground data example $(I, J) \in K$ for which there exists a vfe GAV schema mapping, where K is the GLAV-tractable class of data examples given by Theorem 2.15. Let $\mathbf{S}$ and $\mathbf{T}$ be the source schema and target schema involved. We transform (I, J) to another data example (I', J') over the same source and target schemas: let $E = \{\ (I_1, J_1), \cdots, (I_n, J_n)\ \}$, where each (I_k, J_k) is an isomophic copy of (I, J) using fresh constants, and where $n = 3 \cdot p(||(I, J)||^2)$. We define (I', J') to be the data example where $I' = \bigcup_{k=1\ldots n} I_k$ and $J' = \bigcup_{k=1\ldots n} J_k$.

Clearly, the transformation from (I, J) to (I', J') described above is computable in polynomial time. In the remainder of this section, we show that, from a near-optimal GAV schema mapping for (I', J') (in the sense of Theorem 2.14) we can extract, in polynomial time, a near-minimal vfe GAV schema mapping for (I, J) (in the sense of Theorem 2.15).

Lemma 2.19 Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be any connected GAV schema mapping (where $\mathbf{S}$ and $\mathbf{T}$ are the source and target schema of the data example (I, J)). If $\mathcal{M}$ is vfe for (I, J), then $\mathcal{M}$ is vfe for (I', J').

Proof. By invariance under isomorphism, $\mathcal{M}$ is vfe for (I_k, J_k), for all $k \leq n$. Since $\mathbf{T}$ consists of unary relations, it follows that, for every connected GAV constraint $\sigma \in \Sigma$, we in fact have that the left-hand side of σ is connected, and hence, for every homomorphism h from the left-hand side of σ into I', the range of h is, in fact, contained in I_k for some $k \leq n$, and therefore, since $(I_k, J_k) \models \Sigma$, h is a homomorphism from the right-hand side of σ to J_k, which is contained in J. It follows that $\mathcal{M}$ is valid for (I', J'). Moreover, since every fact in J' belongs to J_k for some k and $\mathcal{M}$ is vfe for (I_k, J_k), we have that $\mathcal{M}$ explaines every fact in J'. $\square$

Lemma 2.20 Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be any GAV schema mapping, where $\mathbf{S}$ and $\mathbf{T}$ are the source and target schema of the data example (I, J).

1. If every vfe repair of $\mathcal{M}$ for (I, J) contains at least one constant in (I, J), then $\text{cost}(\mathcal{M}, E, GAV^{=,\neq}) \geq 2n$, where n is the number of data examples in E.
2. If there is a vfe repair of $\mathcal{M}$ for (I, J) that does not contain any constant in (I, J), then there is a subset $\Sigma' \subseteq \Sigma$ such that $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ is a vfe GAV schema mapping for (I, J).

Proof. (1) Since every vfe repair of $\mathcal{M}$ for (I, J) contains at least one constant in (I, J), every vfe repair of $\mathcal{M}$ for (I_k, J_k) also contains at least one constant in (I_k, J_k) , where $k = 1, \cdots, n$ due to the fact that (I_k, J_k) is a copy of (I, J). Let $\mathcal{M}_r$ be a minimal vfe repair of $\mathcal{M}$ for E, by definition of vfe for a set of ground data examples, $\mathcal{M}_r$ is also a valid and fully explaining GAV$^{=,\neq}$ schema mapping for each $(I_k, J_k) \in E$. Therefore, $\mathcal{M}_r$ contains at least one constant from each data example $(I_k, J_k) \in E$. This indicates that $\mathcal{M}_r$ contains at least n distinct constants. By the definition of repair language, every repaired schema mapping containing a constant has size at least 2, hence $\text{cost}(\mathcal{M}, E, GAV^{=,\neq}) \geq 2n$.

(2) Let $\mathcal{M}_r = (\mathbf{S}', \mathbf{T}', \Sigma'')$ be a valid and fully explaining repair of $\mathcal{M} = (\mathbf{S}', \mathbf{T}', \Sigma)$ for (I, J) and that does not contain any constant in (I, J). Consider the following repairs that can appear in $\mathcal{M}_r$.

(i) An equality $(x = c)$, where c is not in (I, J), in the left-hand side of a GAV-constraint t in $\mathcal{M}$. This equality is cancelling t, thus t can be removed from $\mathcal{M}_r$ without changing the validity and explanation of $\mathcal{M}_r$ for (I, J).

(ii) An inequality $(x \neq c)$, where c is not in (I, J), in the left-hand side of a GAV constraint t in $\mathcal{M}$. Clearly, it is a redundant repair and can be removed from t. This implies a contradiction to the fact that $\mathcal{M}_r$ is a minimal, hence cannot be the case.

(iii) A ground fact f such that $f \notin J$. This cannot be the case, because it makes $\mathcal{M}_r$ invalid in (I, J).

Since Case (ii) and Case (iii) cannot arise, the only possible repair is Case (i). If $\mathcal{M}_r$ contains such an equality, then we can remove the constraint that contains such kind of equality, because the repair is cancelling the constraint. Therefore, we can find a subset $\Sigma' \subseteq \Sigma''$ and that is a vfe for (I, J), and, clearly, Σ' is also a subset of Σ. $\square$

Proposition 2.21 Let (I, J) be any data example from the GLAV-tractable class K given by Theorem 2.15, and let (I', J') be as constructed above (that is, by making isomorphic copies of (I, J)). Let $\mathcal{M}$ be any connected GAV schema mapping such that $\text{cost}(\mathcal{M}, (I', J'), GAV^{=,\neq}) \leq p(\text{cost}(\mathcal{M}^{opt}, (I', J'), GAV^{=,\neq}))$, where $\mathcal{M}^{opt}$ is the optimal GAV schema mapping for (I', J'). From $\mathcal{M}$ and (I, J), we can compute in polynomial time a GAV schema mapping $\mathcal{M}'$ such that $size(\mathcal{M}') \leq 2p(size(\mathcal{M}^{min}))$, where $\mathcal{M}^{min}$ is the smallest vfe GAV schema mapping for (I, J).

Proof. By Lemma 2.16 and Lemma 2.17, there is a vfe connected GAV schema mapping for (I, J) of size at most $||(I, J)||^2$. By Lemma 2.19, the same schema mapping is also vfe for (I', J'), and, therefore, $\text{cost}(\mathcal{M}^{opt}, (I', J'), GAV^{=,\neq}) \leq ||(I, J)||^2$. Let $\mathcal{M}_{conn}$ be the connected GAV schema mapping obtained from $\mathcal{M}$ through Lemma 2.18. Then

$$\text{cost}(\mathcal{M}_{conn}, (I', J'), GAV^{=,\neq}) \leq 2\text{cost}(\mathcal{M}, (I', J'), GAV^{=,\neq}) \leq 2p(||(I, J)||^2) .$$

Recall that $n = 3p(||(I, J)||^2)$. By Lemma 2.20, we obtain that $\mathcal{M}_{conn} = (\mathbf{S}, \mathbf{T}, \Sigma)$ "contains" a GAV mapping $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ (with $\Sigma' \subseteq \Sigma$) that is vfe for (I, J). Moreover, $\mathcal{M}'$ is computable in polynomial time due to the GLAV-tractability of K, and

$$size(\mathcal{M}') \leq size(\mathcal{M}_{conn}) \leq 2\text{cost}(\mathcal{M}, (I', J'), GAV^{=,\neq}) \leq 2p(\text{cost}(\mathcal{M}^{opt}, (I', J'), GAV^{=,\neq})).$$

By Lemma 2.17, we may assume without loss of generality that $\mathcal{M}^{min}$ is connected. By Lemma 2.19, then, $\mathcal{M}^{min}$ is vfe for (I', J') and we have that

$$\text{cost}(\mathcal{M}^{opt}, (I', J'), GAV^{=,\neq}) \leq size(\mathcal{M}^{min}).$$

Putting everything together, we have that

$$size(\mathcal{M}') \leq 2p(\text{cost}(\mathcal{M}^{opt}, (I', J'), GAV^{=,\neq})) \leq 2p(size(\mathcal{M}^{min})). \quad \Box$$

Proposition 2.21 together with Theorem 2.15 immediately implies Theorem 2.14.

Extending the proof of Theorem 2.14 for GLAV schema mappings

Here, we show that how to extend the proof of Theorem 2.14 to hold both for GLAV$^{=,\neq}$ schema mappings and for GLAV$^{=}$ schema mappings.

Proposition 2.22 Let (I, J) be a data example over source and target schema $\mathbf{S}$ and $\mathbf{T}$, where $\mathbf{T}$ contains only unary relations. For every GLAV schema mapping $\mathcal{M}$ there is a GAV schema mapping $\mathcal{M}'$, which can be obtained from $\mathcal{M}$ in polynomial time, such that i $\text{cost}(\mathcal{M}', (I, J), GAV^{=,\neq}) = O(\text{cost}(\mathcal{M}, (I, J), GLAV^{=,\neq})^2)$, and

ii $\text{cost}(\mathcal{M}', (I, J), GAV^{=}) = O(\text{cost}(\mathcal{M}, (I, J), GLAV^{=})^2)$

Proof. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$. We denote by $\mathcal{M}^{\text{gav}}$ the GAV mapping $(\mathbf{S}, \mathbf{T}, \bigcup_{\sigma \in \Sigma} \Sigma^{\text{gav}})$, where Σ^{gav} is the set of GAV constraints obtained by removing all atoms containing existentially quantified variables, as well as all existential quantifiers, from the right-hand side of σ, and, in case of multiple atoms in the right-hand side, splitting the constraint into several constraints, each having a single atom in the right-hand side. For example, if σ is the GLAV constraint $\forall xy(P(x) \land Q(y) \to \exists z(R(x) \land S(y) \land T(z) \land U(z))$, then Σ^{gav} consists of the GAV constraints $\forall xy(P(x) \land Q(y) \to R(x))$ and $\forall xy(P(x) \land Q(y) \to S(y))$.

We claim that $\mathcal{M}^{\text{gav}}$ satisfies the properties. We describe the argument for (i), but the argument for (ii) is similar. Consider any GLAV$^{=,\neq}$ repair $\widehat{\mathcal{M}}$ of $\mathcal{M}$ that is vfe for (I, J). We construct a corresponding GAV$^{=,\neq}$ repair $\widehat{\mathcal{M}^{\text{gav}}}$ for $\mathcal{M}^{\text{gav}}$ as follows:

(a) whenever $\widehat{\mathcal{M}}$ contains a repaired $\widehat{\sigma}$ of a constraint $\sigma \in \Sigma$, then we add to $\widehat{\mathcal{M}^{\text{gav}}}$ a set $\widehat{\Sigma^{\text{gav}}}$ consisting of the repair of each constraint in Σ^{gav} obtained by applying the same equalities and inequalities that involve universally quantified variables.

(b) whenever a $\widehat{\mathcal{M}}$ contains a repaired constraint $\widehat{\sigma}$ containing a conditional equality of the form $(\ldots \to y = c)$, where y is an existentially quantified variable, then for each atom $T(y)$ in the right-hand side of $\widehat{\sigma}$, we add the ground fact $T(c)$ to $\widehat{\mathcal{M}^{\text{gav}}}$.

(c) whenever there is a ground fact f in $\widehat{\mathcal{M}}$, we add f to $\widehat{\mathcal{M}^{\text{gav}}}$

We next show that $\widehat{\mathcal{M}^{\text{gav}}}$ is vfe for (I, J) and it is only quadratically larger than $\widehat{\mathcal{M}}$.

Validity. By construction, the ground facts included in $\widehat{\mathcal{M}^{\text{gav}}}$ are valid for

(I, J), and it suffices to show that the GAV constraints of $\widehat{\mathcal{M}^{\text{gav}}}$ are also valid for (I, J). To show that, we take an arbitrary homomorphism h from the left-hand side of some GAV constraint in $\widehat{\mathcal{M}^{\text{gav}}}$ to I. By construction of $\widehat{\mathcal{M}^{\text{gav}}}$, we have that h is also a homomorphism from the left-hand side of some constraint in $\widehat{\mathcal{M}}$ to I (for every constraint $t^{\text{gav}} \in \widehat{\mathcal{M}^{\text{gav}}}$, we can find a constraint $t \in \widehat{\mathcal{M}}$ that has identical left-hand side as t^{gav}). Since $\widehat{\mathcal{M}}$ is valid for (I, J), we know that h can be extended to a homomorphism h' from the right-hand side of $\widehat{\mathcal{M}}$ to J. We can easily extract a homomorphism h'' from h' such that h'' is a homomorphism from the right-hand side of $\widehat{\mathcal{M}^{\text{gav}}}$ to J (where h'' is the restriction of h' to the universally quantified variables). Therefore, $\widehat{\mathcal{M}^{\text{gav}}}$ is valid for (I, J).

Fully Explaining. Since $\widehat{\mathcal{M}}$ is fully explaining for (I, J), we have that each fact in J is either one of the ground target facts of $\widehat{\mathcal{M}}$ or is explained by some constraint in $\widehat{\mathcal{M}}$. For those facts in J that are among the ground target facts in $\widehat{\mathcal{M}}$, they will also belong to $\widehat{\mathcal{M}^{\text{gav}}}$ (because of Procedure (c) described above). Now, consider a (unary) target fact $P(a)$ that is explained by a constraint $\widehat{\sigma} \in \widehat{\mathcal{M}}$. Intuitively, this means that, if we chase I with respect to $\widehat{\sigma}$, the fact $P(a)$ belongs to the chase result — see [29] for more details about chase procedure). Let σ' be the constraint obtained from σ by dropping all atoms containing existentially quantified variables, and let $\widehat{\sigma'}$ be the repair of σ' obtained from $\widehat{\sigma}$ in the same way. Then either $P(a)$ is already explained by $\widehat{\sigma'}$, or, otherwise, the right-hand side of $\widehat{\sigma}$ must contain an atom of the form $P(y)$ and a conditional equality of the form $(\cdots \to y = a)$. In either case, it follows from our construction that $P(a)$ is explained also by $\widehat{\mathcal{M}^{\text{gav}}}$.

We now show that the size of $\widehat{\mathcal{M}^{\text{gav}}}$ is bounded quadratically in the size of $\widehat{\mathcal{M}}$. By construction, the operations described in procedure (c) will never result in a schema mapping that has greater size than the original one, and the procedure described in (b) will blows up the size by a constant factor. This is because each conditional equality explains at most one ground target fact (note that the target schema contains unary relations only), and it has size at least two). However, the operations described in procedure (a) may blow up the size to a quadratic factor. Therefore, the size of $\widehat{\mathcal{M}^{\text{gav}}}$ is bounded quadratically in the size of $\widehat{\mathcal{M}}$, and the proposition is proved. $\square$

Proposition 2.22, intuitively, shows that the problem of computing near-optimal GAV schema mappings (for unary target schemas) reduces to the problem of computing near optimal GLAV schema mappings. Since the proof of Theorem 2.14 uses a unary target schema, it immediately implies that Theorem 2.14 holds true where GAV is replaced by GLAV.

2.3.2 Complexity of the problems Cost and Existence-Cost for LAV and SH-LAV schema mappings

Next, we show that the problems Cost and Existence-Cost are NP-complete for the languages of LAV schema mappings and SH-LAV schema mappings. Neither of these schema-mapping languages was considered in [38].

Although restrictive, SH-LAV schema mappings together with the SH-LAV$^=$ repair language, can capture projection, column addition, column reordering, and selection conditions. Many of the primitives of two well-known primitive-based schema mapping benchmarks (STBenchmark [2] and iBench [8]) can be expressed as SH-LAV schema mappings and SH-LAV$^=$ repairs. In particular, three out of eleven primitives of STBenchmark correspond to SH-LAV schema mappings and SH-LAV$^=$-repair: Copy, Constant Value Generation, Horizontal Partition. In iBench, there are five out of fifteen: Copy, Horizontal Partition, Add Attribute, Delete Attribute, and Add & Delete Attribute.

In addition, SH-LAV constraints capture a large fragment of the DL-Lite language used in ontology based data access (OBDA) [10]. More precisely, as pointed out in [22], a knowledge base (as defined in the OBDA literature) can be captured, in a precise formal sense, by a schema mapping that is specified by source-to-target copy-constraints as well as target constraints. If the knowledge base is specified in the core DL-Lite language, it turns out that each target constraint is either a SH-LAV constraint or a denial constraint (i.e., a first-order sentence of the form $\forall \mathbf{x}(\phi \rightarrow \bot)$ where ϕ is a conjunction of atoms).

Theorem 2.23 Let $\mathcal{L}^* \in \{\text{LAV}^{=,\neq}, \text{LAV}^{=}, \text{SH-LAV}^{=,\neq}, \text{SH-LAV}^{=}\}$. Then the problems $\textsc{Cost}_{\mathcal{L}^*}$ and $\textsc{Existence-Cost}_{\mathcal{L}^*}$ are NP-complete. In fact, there are fixed schemas $\mathbf{S}$ and $\mathbf{T}$ for which these problems are NP-complete.

Theorem 2.23 is established via a reduction from the Set-Cover problem. The following lemmas and propositions will be helpful for establishing the NP-upper bounds.

Lemma 2.24 Let E be a finite set of ground data examples and $\mathcal{M}$ a LAV schema mapping with $\text{cost}(\mathcal{M}, E, \text{LAV}^{=,\neq}) = n$. Then there is a SH-LAV schema mapping $\mathcal{M}'$ such that $\text{cost}(\mathcal{M}', E, \text{SH-LAV}^{=,\neq}) \leq n^2$. The same holds for when $=, \neq$ is replaced by $=$.

Proof. Let E be a finite set of ground data examples, let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a LAV-schema mapping, and let $\mathcal{M}_r = (\mathbf{S}, \mathbf{T}, \Sigma_r)$ be a minimal-size vfe LAV*-repair of $\mathcal{M}$ for E, where $*$ is either $=$ or $=, \neq$.

We will denote by $\mathcal{M}_r^{\text{shlav}}$ the SH-LAV*-schema mapping $(\mathbf{S}, \mathbf{T}, \Sigma_r^{\text{shlav}})$ where Σ_r^{shlav} is obtained as follows: for each LAV*-constraint $\sigma \in \Sigma$ of the form

$$R(\mathbf{x}) \wedge \alpha(\mathbf{x}) \rightarrow \exists \mathbf{y}(S_1(\mathbf{x}, \mathbf{y}_1) \wedge \cdots \wedge S_n(\mathbf{x}, \mathbf{y}_n) \wedge \beta(\mathbf{x}, \mathbf{y})) \ ,$$

(with $\mathbf{y}_1, \ldots, \mathbf{y}_n \subseteq \mathbf{y}$) we take all n SH-LAV* constraints σ' of the form

$$R(\mathbf{x}) \wedge \alpha(\mathbf{x}) \rightarrow \exists \mathbf{y}_i(S_i(\mathbf{x}, \mathbf{y}_i) \wedge \beta'(\mathbf{x}, \mathbf{y}_i)) \ ,$$

where $1 \leq i \leq n$, and where β' is obtained from β by keeping only those conjuncts that contain an existentially quantified variable belonging to $\mathbf{y}_i$. Thus, for example, if $\mathcal{M}_r$ contains the LAV$^=$-constraint $R(x, y) \rightarrow \exists uv\ S(x, u) \wedge T(z, u, v) \wedge (x = c_1 \wedge y = c_2 \rightarrow u = c_3) \wedge (x = c_1 \rightarrow v = c_4)$, then $\mathcal{M}_r^{\text{shlav}}$ contains $R(x, y) \rightarrow \exists u\ S(x, u) \wedge (x = c_1 \wedge y = c_2 \rightarrow u = c_3)$ and $R(x, y) \rightarrow \exists uv\ T(z, u, v) \wedge (x = c_1 \wedge y = c_2 \rightarrow z = c_3) \wedge (x = c_1 \rightarrow v = c_4)$.

We will denote by $\mathcal{M}_r^{\text{shlav}}$ the SH-LAV schema mapping obtained in the same way from $\mathcal{M}$. Observe that $\mathcal{M}_r^{\text{shlav}}$ is a SH-LAV*-repair of $\mathcal{M}^{\text{shlav}}$, and that $size(\mathcal{M}_r^{\text{shlav}}) \leq size(\mathcal{M}_r^2)$. It only remains to show that $\mathcal{M}_r^{\text{shlav}}$ is vfe for E. Validity is immediate from the construction (note that $\mathcal{M}_r$ logically implies $\mathcal{M}_r^{\text{shlav}}$. It is also clear from the construction that every fact in the target instance of a data example from E

explained by a LAV*-constraint σ of $\mathcal{M}_r$ is also explained by one of the corresponding SH-LAV*-constraints of $\mathcal{M}_r^{\text{shlav}}$. $\square$

Definition 2.25 (csize(E)) The constant-based size of a set E of ground data examples, denote by csize(E), is the total number of occurrences of constants in all data examples in E.

Note that the quantity csize(E) is different from the quantity $||E||$ introduced in Section 2.1: the former gives the total number of occurrences of constants in E, while the latter gives the total number of facts in E. We use $||E||$ as an amplification parameter in Lemma 2.16, which is part of the proof of Theorem 2.14. In the next lemma, we will use csize(E) to show that when a vfe SH-LAV$^=$-schema mapping for E exists, then there is a "small" one.

Lemma 2.26 Let E be a finite set of ground data examples for which there exists a vfe SH-LAV$^=$-schema mapping. There exists a SH-LAV schema mapping $\mathcal{M}$ such that $\text{cost}(\mathcal{M}, E, \text{SH-LAV}^=)$ is bounded by a polynomial in csize(E).

Proof. We know that there is a vfe SH-LAV$^=$-schema mapping for E. Each constraint t in a minimal vfe SH-LAV$^=$-schema mapping can explain at least one target fact (because otherwise, we can remove all non-explaining constraints and that will still be vfe for E but has smaller size). In the worst case, suppose that each SH-LAV$^=$ constraint in a minimal vfe SH-LAV$^=$-schema mapping only explains one target fact in E. Let $\mathcal{M}$ be a minimal vfe SH-LAV$^=$-schema mapping, let n be csize(E). The arity of atoms occurring in a SH-LAV$^{=}$-constraint $t \in \mathcal{M}$ are both bounded by n, because $\mathcal{M}$ is minimal vfe for E.

Each variable occurring in the left-hand side of t can only be involved in at most one equality, because, if the left-hand side of t two equalities involving same universally quantified variable but using different constants, then this constraint is trivial and can be removed. Also, in the worst case that all variables occurring in the right-hand side are existentially quantified, we have to add repairs that involve every variable in the right-hand side. The most costly way to repair an existentially quantified variable is to add a conditional equality with a conjunction of n equalities in the premise, and thus the maximum size of a repair on the right-hand side is $n \times (2n + 1)$. (Note that in fact the same existentially quantified variable can be repaired by multiple conditional equalities. However, it is enough to explain one fact with the repaired constraints with one conditional equality per existentially quantified variable). Therefore, the maximum size of an explaining SH-LAV$^{=}$ constraint for E is $2n^2 + 5n$ (n for unrepaired left-hand side, n for unrepaired right-hand side, $2n$ for the equalities on the left-hand side, and $2n^2 + n$ for the conditional equalities on the right-hand side). Since the number of facts in E is at most n, the maximum size of a minimal vfe SH-LAV$^{=}$ schema mapping for E is at most $2n^3 + 5n^2$, therefore polynomial in csize(E). $\square$

Proposition 2.27 Fix a source $\mathbf{S}$ and target schema $\mathbf{T}$. Let E be a finite set of ground data examples over $\mathbf{S}$ and $\mathbf{T}$. If there exists a vfe LAV$^{=,\neq}$-schema mapping for E, then there is a vfe LAV$^{=,\neq}$-schema mapping (in fact, a SH-LAV$^{=}$-schema mapping) $\mathcal{M}$ for E whose size is bounded by a polynomial in $csize(E)$.

Proof. Lemma 2.24 shows that if a vfe LAV$^{=,\neq}$-schema mapping for E exists with cost n, then a vfe SH-LAV$^{=,\neq}$-schema mapping, whose cost is bounded by n^2, for E exists. To prove the Proposition, we first show that for every vfe SH-LAV$^{=,\neq}$-schema mapping $\mathcal{M}$ for E, we can efficiently (in polynomial time) rewrite $\mathcal{M}$ into a SH-LAV$^{=}$-schema mapping $\mathcal{M}'$ such that $\mathcal{M}$ and $\mathcal{M}'$ are logically equivalent. This will show the existence of a vfe SH-LAV$^{=}$-schema mapping for E. Then, by Lemma 2.26, we know that there must be a SH-LAV$^{=}$-schema mapping for E whose size is polynomial in $csize(E)$.

Let $\mathcal{M}$ be the SH-LAV$^{=,\neq}$-schema mapping that is vfe for E, and assume that every constraint $t \in \mathcal{M}$ is of the form:

$$S(\mathbf{x}) \wedge \alpha(\mathbf{x}) \wedge \beta(\mathbf{x}) \rightarrow \exists \mathbf{y} T(\mathbf{x}, \mathbf{y}) \wedge \theta(\mathbf{x}, \mathbf{y}), \text{ where}$$

$\alpha(\mathbf{x})$ is a collection of equalities with each of the form $(x = c)$,

$\beta(\mathbf{x})$ is a collection of inequalities with each of the form $(x \neq c)$.

and $\theta(\mathbf{x}, \mathbf{y})$ is a collection of conditional equalities with each of the form $\bigwedge_i x_i = c_i \rightarrow y_k = c_k)$. (Note that $x_i \in \mathbf{x}, y_i \in \mathbf{y}$, and c, c_i, c_k are constants.)

Let $adom(E)$ be the union of the active domain of all source and target instances in E. We construct a SH-LAV$^{=}$-schema mapping $\mathcal{M}'$ from $\mathcal{M}$ such that $\mathcal{M}'$ is the schema mapping that contains, for each constraint $t \in \mathcal{M}$, all constraints $t\prime$ that can be obtained from t by replacing each inequality $x \neq c$ by an equality $x = d$ for some constant $d \in adom(E)$ such that $d \neq c$, provided that t' does not contain multiple equalities $x = d_1$ and $x = d_2$ for the same universally quantified variable, with $d_1 \neq d_2$. As an illustrative example, assume that we have a ground data example (I, J), where

$I = \{S(a),\ S(b),\ S(c)\}$

$J = \{R(a),\ R(b)\}$,

and we are given following SH-LAV$^{=,\neq}$-schema mapping that is vfe for (I, J)

$$\mathcal{M} = \big(\{S\}, \{R\}, \Sigma = \{S(x) \wedge (x \neq c) \rightarrow R(x)\}\big).$$

Then we can obtain a new schema mapping $\mathcal{M}'$ by rewriting Σ into set Σ' of constraints (see below):

$$\mathcal{M}' = \big(\{S\}, \{R\}, \Sigma' = \{S(x) \wedge (x = a) \rightarrow R(x),\ S(x) \wedge (x = b) \rightarrow R(x)\big)$$

In this way, the newly constructed schema mapping $\mathcal{M}'$ has the same validity and explanation properties with respect to E as the original schema mapping $\mathcal{M}$. Moreover, since the number of universally quantified variables in a constraint is bounded by a constant (i.e., the arity of the source schema) the number of constraints in $\mathcal{M}'$ is polynomial in the number of constraints of $\mathcal{M}$.

Note that a SH-LAV$^{=}$-schema mapping is also a SH-LAV$^{=,\neq}$-schema mapping and a LAV$^{=,\neq}$-schema mapping. Moreover, for any schema mapping language $\mathcal{L}$, given a finite set E of ground data examples, the size of smallest vfe $\mathcal{L}^{=,\neq}$-schema mapping for E is no greater than the size of smallest vfe $\mathcal{L}^{=}$-schema mapping. Therefore, the above two lemmas (Lemmas 2.24 and 2.26) imply Proposition 2.27. $\square$

Before presenting the reduction from Set-Cover problem to our problem, we need to show that our problem is in NP and the following lemma serves the purpose.

Lemma 2.28 Given a finite set E of ground data examples and a LAV$^{=,\neq}$ schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, determining whether $\mathcal{M}$ is vfe for E is in NP. If $\mathcal{M}$ is a SH-LAV$^{=,\neq}$ schema mapping, the same problem is in P.

Proof. Since $\mathcal{M}$ is LAV$^{=,\neq}$, each constraint in Σ is of the form

$$\forall \mathbf{x}(S(\mathbf{x}) \wedge \alpha(\mathbf{x}) \rightarrow \exists \mathbf{y}(T_1(\mathbf{x}_1, \mathbf{y}_1) \wedge \cdots \wedge T_m(\mathbf{x}_m, \mathbf{y}_m) \wedge \beta(\mathbf{x}, \mathbf{y})))$$

where S($\mathbf{x}$) is a relational atom over source schema, $\alpha(\mathbf{x})$ is a collection of equalities with each of the form $(x = c)$, $\beta(\mathbf{x}, \mathbf{y})$ is a collection of conditional equalities with each of the form $(\bigwedge_i x_i = c_i \rightarrow y_k = c_k)$. (Note that $x_i \in \mathbf{x}, y_i \in \mathbf{y}$, and c, c_i, c_k are constants), and T_i($\mathbf{x}_i$,$\mathbf{y}_i$) is a relational atom over target schema. Let n be the combined size of E and $\mathcal{M}$, that is, $n = \text{csize}(E) + size(\mathcal{M})$. For simplicity, we consider the case where Σ contains only one constraint σ. The same arguments apply in the case of a set of constraints.

Validity checking. There are at most n homomorphisms from $S(\mathbf{x})$ (one for each source fact in E) to a source instance I, with $(I, J) \in E$. For every such homomorphism, we can efficiently check if $\alpha(\mathbf{x})$ is satisfied. For each homomorphism h that makes $S(h(\mathbf{x})) \wedge \alpha(h(\mathbf{x}))$ be true, we non-deterministically guess an extension h' of h w.r.t. the existential variables in the right-hand side of σ, such that $\beta(h'(\mathbf{x}), h'(\mathbf{y}))$ holds. We can then verify in polynomial time that h' is a homomorphism from the right-hand side of σ to the target instance J.

Explanation checking. The total number of target facts in the data examples is at most n, and it suffices to check that each target fact is explained by $\mathcal{M}$. Let

$(I, J) \in E$ and let $T(\mathbf{a})$ be a fact belonging to J. To test if $\mathcal{M}$ explains $T(\mathbf{a})$, we guess a homorphisms h from the left-hand side of σ to I such that the right-hand side of σ under h (i.e., $\exists \mathbf{y}(T_1(h(\mathbf{x}_1), \mathbf{y}_1) \wedge \cdots \wedge T_m(h(\mathbf{x}_m), \mathbf{y}_m) \wedge \beta(h(\mathbf{x}), \mathbf{y})))$ logically implies $T(\mathbf{a})$. This can be done in polynomial time (due to the fact that $T(\mathbf{a})$ is a ground atom). If the answer is positive, then, clearly, every solution of I w.r.t. $\mathcal{M}$ (that is, every target instance J such that $\mathcal{M}$ is valid w.r.t. (I, J)) must contain $T(\mathbf{a})$, and hence, $\mathcal{M}$ explains $T(\mathbf{a})$ w.r.t. (I, J). If not, then construct a solution of I with respect to $\mathcal{M}$ that does not contain $T(\mathbf{a})$, and hence, $\mathcal{M}$ does not explain $T(\mathbf{a})$ w.r.t. (I, J).

On the other hand, checking the validity and explanation of SH-LAV$^{=,\neq}$-schema mapping is in P. Because there are at most n homomorphisms from the left-hand side of a SH-LAV$^{=,\neq}$-constraint t to E and for each homomorphism h that makes the left-hand side of t be true, there are also at most n homomorphisms from the right-hand side of a SH-LAV$^{=,\neq}$-constraint to E. We can try then all, and the total computation is polynomial in the size of input. □

We are ready to prove Theorem 2.23.

Proof of Theorem 2.23. We will prove NP-completeness of EXISTENCE-COST$_{\text{LAV}^{=,\neq}}$. The proof of the other items in Theorem 2.23 is identical.

First, we need to show the problem is in NP. Let $\mathbf{S}$ be a source schema, $\mathbf{T}$ be a target schema. Given a finite set E of ground data examples such that E conforms to $\mathbf{S}$ and $\mathbf{T}$. By Proposition 2.27 we know that if there is a vfe LAV$^{=,\neq}$ schema mapping for E, then there is one whose size is bounded by $p(||E||)$, for some polynomial p. This places the problem in NP: we can guess a schema mapping of size at most $p(||E||)$, and verify the guess in polynomial-time with an additional non-deterministic guess using Lemma 2.28. Note that both non-deterministic guesses can be combined into one guess. The NP-hardness is shown via a reduction from SET-COVER [45].

SET-COVER: an instance of the set-cover problem is a pair (X, S) where X is a set and S is a collection of subsets of X, such that $\bigcup_{S_i \in S} S_i = X$. A solution for (X, S) is a subset S' of S such that $\bigcup_{S_k \in S'} S_k = X$. An optimal solution is a solution such that the size of S' is minimum.

The following problem is known to be NP-complete: given an instance of the set-cover problem (X, S) and a natural number K, does there exist a solution S' with $|S'| \leq K$?

We encode an instance (X, S) of the set-cover problem by a data example (I, J) over the schemas $\mathbf{S}, \mathbf{T}$, where $\mathbf{S}$ consists of a single 5-ary relation P, and $\mathbf{T}$ consists of a single 4-ary relation Q. For each element $a \in X$, we create four distinct constants a_1, a_2, a_3, a_4. The source instance I contains all facts $P(i, a_1, a_2, a_3, a_4)$ for $S_i \in S$ and $a \in S_i$ (we assume, without loss of generality, that the index i is not itself an element in X). In addition, I contains $7|X|$ facts of the form $P(b, c, d, e, f)$, where b, c, d, e, f are fresh distinct constants not contained in X. We will refer to these additional facts as *garbage facts* below. The target instance J contains all facts $Q(a_1, a_2, a_3, a_4)$ for $a \in X$. Obviously, the construction of (I, J) from (X, S) is a polynomial-time transformation.

Claim: the optimal solution for (X, S) has size of k iff $\text{cost}(M_{opt}, \{(I, J)\}, LAV^{=,\neq}) = 11k$, where M_{opt} is LAV$^{=,\neq}$-optimal for (I, J)

($\Rightarrow$) Let $S' \subseteq S$ be an optimal solution for the set-cover problem (X, S), and let $|S'| = k$. We will construct a vfe LAV$^{=,\neq}$ schema mapping for (I, J) and that has size $11k$. Consider the LAV schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ where $\Sigma = \{t\}$ for

$$t : P(x_1, y_1, y_2, y_3, y_4) \rightarrow Q(y_1, y_2, y_3, y_4).$$

Let Σ_r be the set containing, for each $S_i \in S'$, the LAV$^{=,\neq}$ constraint

$$t'_i : \ P(x, y_1, y_2, y_3, y_4) \wedge (x = i) \rightarrow Q(y_1, y_2, y_3, y_4).$$

Clearly, $\mathcal{M}_r = (\mathbf{S}, \mathbf{T}, \Sigma_r)$ is a repair of $\mathcal{M}$, and $\mathcal{M}_r$ is vfe for (I, J), and the size of $\mathcal{M}_r$ is $11k$. Now we will show that $\mathcal{M}$ is LAV$^{=,\neq}$-optimal for (I, J), because no other vfe LAV$^{=,\neq}$ schema mapping for (I, J) has smaller size than Σ_r.

Let $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ be any vfe LAV$^{=,\neq}$-schema mapping for (I, J) of size ℓ. We may assume that Σ' does not contain ground facts: every ground fact costs 12 (recall that a ground fact costs $3n$, where n is the arity of the fact), while the same fact can be explained by a valid constraint of the form t'_i above, for suitable choice of i, which costs only 11. Therefore, Σ' consists of LAV$^{=,\neq}$-constraints of the form

$$P(\mathbf{x}) \wedge \alpha(\mathbf{x}) \rightarrow \exists \mathbf{w} Q(\mathbf{x}, \mathbf{w}) \wedge \cdots \wedge Q(\mathbf{x}, \mathbf{w}) \wedge \beta(\mathbf{x}, \mathbf{w}).$$

We may furthermore assume that each constraint in Σ' explains at least one target fact (all constraints that do not explain any target fact can be removed without increasing the size of $\mathcal{M}'$ and the resulting schema mapping is still vfe for (I, J)).

Consider the left-hand side of the above constraint. We claim that all variables occurring in the P-atom are distinct, that is, $P(\vec{x})$ is of the form $P(x, y_1, y_2, y_3, y_4)$:

otherwise, the constraint cannot explain any fact in J. Furthermore, the variable x that is in the *first* position of P-atom does not occur in the right-hand side of the constraint (otherwise either the constraint is invalid for (I, J), or it does not explain any fact), and that the variable y_j (for $1 \leq j \leq 4$) occurs only in the j-th position of atom in the right-hand side of the constraints (again, otherwise the constraint is either invalid or unable to explain a fact).

Now consider the right-hand side of the constraint. We may assume without loss of generality that every Q-atom contains at least one variable that is not existentially quantified. Indeed, if some atom $Q(w_1, \dots, w_4)$ consisted entirely of existentially quantified variables, then, in order for the atom to explain a fact, β must include 4 (possibly conditional) equalities of the form $(w_j = c_j)$ $(1 \leq j \leq 4)$. These equality conditions can be removed at the expense of adding the constraint t'_i (as defined above) for some $S_i \in S$ with $c \in S_i$ (which is valid and explains the same fact, while having a lower cost), after which the atom $Q(w_1, \dots, w_4)$ can be removed from the right-hand side of the constraint entirely.

Thus, each Q-atom contains one of the variables y_j in its j-th position. It follows, through the construction of (I, J) that, under any given assignment of values to the variables $x, y_1, \dots, y_n$ that makes the left-hand side true, each Q-atom in the right-hand side can explain at most one fact of J, namely the fact that has the value of y_j in its j-th position. But this means that any Q-atom of the form $Q(\dots, y_j, \dots)$ can be replaced by $Q(y_1, \dots, y_n)$ without changing the collection of facts explained by the constraint, or the validity of the constraint, and without increasing the size of the constraint. This also shows that there is no benefit in having more than one Q-atom in the right-hand side of the constraint. To summarize, from the assumption that $\mathcal{M}'$ is a minimal vfe LAV$^{=,\neq}$-schema mapping for (I, J), we have derived that every constraint of $\mathcal{M}'$ is of the form

$$P(x, y_1, y_2, y_3, y_4) \wedge \alpha(x, \mathbf{y}) \rightarrow Q(y_1, y_2, y_3, y_4)$$

Finally, we argue that, without loss of generality, $\alpha(x, \mathbf{y})$ is of the form $(x = i)$. If $\alpha(\mathbf{x})$ contains an equality of the from $y_j = c$ we can replace it by an equality of the form $x = i$ for any i such that $c \in S_i$, without affecting validity and the resulting constraint explains all the same facts and possibly more. If α contains an inequality $z \neq c$, then, it follows from the construction of the data example (I, J) that either the inequality can be removed without affecting the validity of the constraint, or the left-hand-side must contain as many inequalities as the number of garbage facts, in which case the size of the constraint is at least $14|X|$ (each inequality has size 2). In the latter case, we can replace the schema mapping by one that consists of $|X|$ many constraints of the form t'_i — at most one for each element of X in the worst case, with a total cost of at most $11|X|$.

In summary, we may assume without loss of generality that the constraint is of the form

$$P(x, y_1, y_2, y_3, y_4) \wedge (x = i) \rightarrow Q(y_1, y_2, y_3, y_4)$$

and indeed, by the above arguments, every minimal-size vfe LAV$^{=,\neq}$-schema mapping for (I, J) must consist entirely of constraints of the above form. The collection of target facts explained by this constraint corresponds precisely to the the set S_i from the set-cover problem. In this way, we obtain that every minimal-size vfe LAV$^{=,\neq}$-schema mapping for (I, J) has cost $11k$ for some k, and induces a solution of the set-cover problem (X, S) that has size k.

($\Leftarrow$) Suppose the cost of a LAV$^{=,\neq}$-optimal schema mapping $\mathcal{M}$ for (I, J) is $11k$. Let $\mathcal{M}_r$ be the minimal vfe $LAV^{=,\neq}$-repair of $\mathcal{M}$. By the above arguments, we know that each constraint of $\mathcal{M}_r$ is of the form $P(x, y_1, \dots, y_n) \wedge (x = c) \rightarrow Q(y_1, \dots, y_n)$. But then, by the above arguments, we have that the set-cover problem (X, S) admits a solution of size k, and, moreover, we have that the set-cover problem admits no smaller solution (as such as smaller solution would entail a smaller vfe LAV$^{=,\neq}$ for (I, J).

The NP-completeness of the Cost problems now follows, since Set-Cover is NP-hard. $\square$

2.4 Approximation of Optimal Single-Head LAV Mappings

We now study the approximation properties of the following optimization problem:

Definition 2.29 The Optimal-Repair$_{\mathcal{L}^*}(\mathbf{S}, \mathbf{T})$ problem asks: given a set E of ground data examples with source schema $\mathbf{S}$ and target schema $\mathbf{T}$, compute a minimal-size valid and fully explaining $\mathcal{L}^*$-schema mapping for E.

Note that, in the above formulation of the problem, the source and target schemas are fixed. One may also consider a variant where the schemas are part of the input. We will not consider the latter here, and we leave its complexity as an open problem to be addressed in future work. We note, though, that the difference between these two variants (i.e., the variant in which the source and target schemas are fixed and the variant in which they are part of the input) is, in some sense, the difference between *data complexity* and *expression complexity*; for this reason, we expect that the second variant will be of higher complexity than the first.

The above problem is equivalent to the problem that asks to compute an optimal $\mathcal{L}$-schema mapping $\mathcal{M}$ together with a minimal-size $\mathcal{L}^*$-repair of $\mathcal{M}$; in particular, it contains, as a special case, the problem of computing an optimal $\mathcal{L}$-schema mapping for a given ground data example. This is so because, from a minimal-size valid and fully explaining $\mathcal{L}^*$-schema mapping, we can immediately extract an optimal $\mathcal{L}$-schema mapping by dropping all equalities, inequalities, and ground facts. Therefore, it follows from Theorem 2.14 that (assuming RP $\neq$ NP) there is no polynomial-time algorithm that solves $\textsc{Optimal-Repair}_{\mathcal{L}^*}$, when $\mathcal{L}^* \in \{\mathrm{GLAV}^{=}, \mathrm{GLAV}^{=,\neq}, \mathrm{GAV}^{=}, \mathrm{GAV}^{=,\neq}\}$.

We establish a positive approximability result for SH-LAV$^{=}$ mappings.

Theorem 2.30 Fix a pair of schemas $\mathbf{S}, \mathbf{T}$. There is a polynomial-time $\mathcal{H}(n)$-approximation algorithm for $\textsc{Optimal-Repair}_{\text{SH-LAV}^{=}}(\mathbf{S}, \mathbf{T})$, where $\mathcal{H}(n) = \sum_{i=1}^{n} \frac{1}{i}$ is the n-th harmonic number.

Our approximation algorithm, see Algorithm GreedySHLAV$^{=}(\mathbf{S}, \mathbf{T})$ shown in Figure 2.1, is obtained by establishing a close connection between $\textsc{Optimal-Repair}_{\mathcal{L}^*}$ and $\textsc{Set-Cover}$. It is known that, although the $\textsc{Set-Cover}$ problem is not constant-

```
Input: finite set $E$ of data examples
Output: a SH-LAV$^=$-schema mapping that is vfe for $E$ or "None exists"
1 Candidates $\leftarrow$ all SH-LAV$^=$ constraints over $\mathbf{S}, \mathbf{T}$ (up to variable renaming) that are
  valid for $E$, plus all ground target facts that belong to every data example in $E$.
2 Unexplained $\leftarrow$ all triples $(I, J, F)$ with $(I, J) \in E$ and $F$ a ground fact belonging to $J$.
  $M \leftarrow \emptyset$
  while Unexplained $\neq \emptyset$ do
3     if Candidates=$\emptyset$ then
4         return "No valid and fully explaining SH-LAV$^=$-schema mapping exists"
5     end
6     Find a $t \in$ Candidates that minimizes

          $\alpha = \frac{size(t)}{|\{(I,J,F) \mid (I,J) \in E \text{ and } F \in Facts_{(I,J)}(t)\} \cap \text{Unexplained}|}$

      $M \leftarrow M \cup \{t\}$
      Unexplained $\leftarrow$ Unexplained $\setminus \{(I,J,F) \mid (I,J) \in E$ and $F \in Facts_{(I,J)}(t)\}$
      Candidates $\leftarrow$ Candidates $\setminus \{t\}$
7 end
```

Figure 2.1: Algorithm GreedySHLAV$^=$($\mathbf{S}, \mathbf{T}$)

approximable, it is $\mathcal{H}(n)$-approximable, where n is the size of the universe. Moreover, this approximation ratio is known to be asymptotically optimal: unless P = NP, SET-COVER can only be approximated up to a factor of $c\ln(n)$, where c is a constant (specified in [4, 44]) and n is the size of universe (recall that $\mathcal{H}(n) = O(\ln n)$). Our approximation algorithm is largely based on the approximation algorithm for WEIGHTED SET-COVER [27]. Here, we can think of each constraint as describing a set of facts, namely, the set of target facts that it explains, and the size of the constraint is the weight of the corresponding set.

The above approximation algorithm can be extended in a straightforward manner to the case of SH-LAV$^{=,\neq}$-constraints with a bounded number of inequalities per variable, as well as to LAV$^{=,\neq}$-constraints with a bounded number of inequalities per variable and a bounded number of atoms in the right-hand side. We leave it as an open problem whether an analog of Theorem 2.30 holds for the general case of SH-LAV$^{=,\neq}$ and LAV$^{=,\neq}$.

For a data example (I, J) and a SH-LAV$^{=}$ constraint t, we denote by $Facts_{(I,J)}(t)$ the set of ground target facts that are explained by t in (I, J). For a ground target fact t, $Facts_{(I,J)}(t) = \{t\}$ if $t \in J$ and $Facts_{(I,J)}(t) = \emptyset$ otherwise. Our algorithm, called GreedySHLAV$^{=}$ is given in Figure 2.1.

The Algorithm GreedySHLAV$^{=}(\mathbf{S}, \mathbf{T})$ is established based on a known greedy algorithm for the Weighted Set-Cover problem. The set of all target facts occurring in a given finite set E of data examples can be seen as the universe of elements that needs to be covered (explained), and each valid SH-LAV$^{=}$-constraint can be seen as inducing a subset that covers a number of elements (explained target facts) of the universe, and the size of such a SH-LAV$^{=}$-constraint can be seen as the cost associated to the corresponding subset. Specifically, Line 1 initializes the variable "Candidates" that contains all possible SH-LAV$^{=}$ constraints that are valid for E over $\mathbf{S}$ and $\mathbf{T}$, and this corresponds to creating the collection of "subsets" of the Weighted Set-Cover instance. Line 2 initializes the variable "Unexplained" that corresponds to the "universe" of the Weighted Set-Cover instance for which we are trying to find a cover. The while loop in the algorithm is precisely the greedy algorithm for Weighted Set-Cover. In particular, at each iteration, the algorithm will pick a constraint (corresponds to a subset) that has the highest cost effectiveness (which is defined on Line 8). The following theorem formally shows that this algorithm is a polynomial-time $\mathcal{H}(n)$-approximation algorithm for our problem.

Theorem 2.31 For each fixed pair of schemas $\mathbf{S}, \mathbf{T}$, GreedySHLAV$^{=}(\mathbf{S}, \mathbf{T})$ is a polynomial time $\mathcal{H}(n)$-approximation algorithm for $\textsc{Optimal-Repair}_{\text{SH-LAV}^{=}}(\mathbf{S}, \mathbf{T})$.

We first introduce the following helper lemma for the proof of Theorem 2.31

Lemma 2.32 Let E be a finite set of ground data examples with fixed source schema $\mathbf{S} = \{S_1, \cdots, S_p\}$ and target schema $\mathbf{T} = \{T_1, \cdots, T_q\}$. Let m be the maximum arity of source relations and n the maximum arity of target relations, and let $|adom(E)|$ be the size of active domain of E. There are at most $pqm^m(m+n)^n(|adom(E)|^m n)^n$ distinct SH-LAV$^{=}$-constraints for E, up to renaming of variables.

Proof. Note that p, q, m, n are constants since schemas are fixed. Observe that there are at most $pqm^m(m+n)^n$ distinct SH-LAV constraints over $\mathbf{S}, \mathbf{T}$, up to renaming of variables. In particular, we can decompose the number as follows.

(1) $p \times q$ - maximum of combinations of source relation and target relation.

(2) m^m - given a source relation S_k in $\mathbf{S}$, the maximum number of distinct single relational atoms over S_k.

(3) $(m+n)^m$ - given a source relation $S_k \in \mathbf{S}$ and a target relation $T_k \in \mathbf{T}$, the maximum number of distinct single relational atoms over T_k. In particular $(m+n)$ is the maximum number of variables, both universally quantified and existentially quantified, that can be used for constructing an atom over T_k.

Since only equalities are allowed in the repairs, thus for each SH-LAV constraint there are at most $|adom(E)|^m$ distinct repairs on the left-hand side, and at most n existentially quantified variables on the right-hand side. For each existentially quantified variable y_k on the right-hand side, there is possibly a conditional equality. It follows that we have at most $|adom(E)|^m$ distinct ways to construct the premise of a conditional equality, and at most n ways to construct the conclusion of the corresponding equality. It follws that there are at most $(|adom(E)|^m n)^n$ ways to construct the right-hand side. Therefore, there are at most $pqm^m(m+n)^n(|adom(E)|^m n)^n$ distinct SH-LAV$^=$-constraints over $\mathbf{S}$, $\mathbf{T}$, and E. $\square$

We are now ready to prove Theorem 2.31.

Proof of Theorem 2.31. The GreedySHLAV$^=(\mathbf{S},\mathbf{T})$ algorithm is derived from the $\mathcal{H}(n)$-approximation algorithm for Weighted Set-Cover problem. Indeed, the problem of finding a minimal-size vfe SH-LAV$^=$-schema mapping can be cast as a weighted set-cover problem: the set of all target facts of a given finite set E of data examples can be seen as the universe of elements that needs to be covered (explained), and each valid and explaining SH-LAV$^=$-constraint can be seen as a subset that covers (explain) a number of elements (target facts) of the universe, and the size of such a SH-LAV$^=$-constraint can be seen as the cost associated to the corresponding subset. Note that the optimal SH-LAV$^=$-schema mapping for a finite set E of data examples, if exists, must be a subset of the set of all valid and explaining SH-LAV$^=$-constraints over $\mathbf{S}$, $\mathbf{T}$, E, plus all ground facts of E. We can view the GreedySHLAV$^=(\mathbf{S},\mathbf{T})$ algorithm as a special case of the well-known greedy $\mathcal{H}(n)$-approximation algorithm for Weighted Set-Cover problem. In particular, the correctness of our algorithm follows from the correctness of the aforementioned approximation algorithm for Weighted Set-Cover [27].

It remains to show that the algorithm runs in polynomial time. Since the schema mappings in consideration are SH-LAV$^=$, by Lemma 2.28, checking validity and explanation of a SH-LAV$^=$-schema mapping for a finite set E of ground data examples can be done in polynomial time. Moreover, since the source schema and target schema in question are fixed, it suffices to show that the number of SH-LAV$^=$-constraints is polynomial in the size of E. By Lemma 2.32, this is indeed the case. Therefore, GreedySHLAV$^=(\mathbf{S},\mathbf{T})$ runs in polynomial time and it is a poly-time $\mathcal{H}(n)$-approximation algorithm for Optimal-Repair$_{\text{SH-LAV}^=}(\mathbf{S},\mathbf{T})$. □

In Section 2.5.1, we will discuss practical implementation details for the approximation algorithm.

2.4.1 A Matching Lower Bound

We conclude with a matching lower bound for the approximability of Optimal-Repair$_{\text{SH-LAV}^=}(\mathbf{S},\mathbf{T})$. This result is established via an approximation-preserving reduction (more precisely, an L-reduction [58]) from Minimum Set Cover.

Theorem 2.33 Let $\mathcal{L} \in \{\text{LAV},\text{SH-LAV}\}$. There are fixed schemas $\mathbf{S}$ and $\mathbf{T}$, and a constant c such that there is no polynomial-time $c\ln(n)$-approximation algorithm for Optimal-Repair$_{\mathcal{L}^=}(\mathbf{S},\mathbf{T})$, where n is the total number of target facts in the input data example. The same holds true for Optimal-Repair$_{\mathcal{L}^{=,\neq}}(\mathbf{S},\mathbf{T})$.

We first recall the definition of L-reduction. Let Π, Π' be two optimization problems, we say Π L-reduces to Π', denoted by $\Pi \preceq_L \Pi'$, if there is a pair of polynomial-time algorithms f, g, and constants $\alpha, \beta > 0$ such that for each instance Ins of Π, the following hold:

(a) Algorithm f produces an instance $Ins' = f(Ins)$ of Π' such that $OPT(Ins') \leq \alpha \cdot OPT(Ins)$, where $OPT(Ins')$ is the cost of an optimal solution for Ins' and $OPT(Ins)$ is the cost of an optimal solution for Ins.

(b) Given any solution of Ins' with cost c', algorithm g produces a solution of Ins with cost c such that $|c - OPT(Ins)| \leq \beta |c' - OPT(Ins')|$.

To prove Theorem 2.33, it suffices to prove the following theorem.

Theorem 2.34 There are L-reductions from MINIMUM SET-COVER to OPTIMAL-REPAIR$_{\mathcal{L}^{=}}(\mathbf{S}, \mathbf{T})$ and to OPTIMAL-REPAIR$_{\mathcal{L}^{=,\neq}}(\mathbf{S}, \mathbf{T})$, where $\mathcal{L} \in \{$LAV, SH-LAV$\}$.

Proof. We spell out the proof for the LAV$^{=,\neq}$ case. The proof can be adapted to obtain also the L-reductions to $\mathcal{L}^{=}$ and $\mathcal{L}^{=,\neq}$, where $\mathcal{L} \in \{\text{LAV}, \text{SH-LAV}\}$.

Let f be the transformation function used in the reduction of the proof of Theorem 2.23. f is a poly-time computable function that transforms an instance Ins of MIN SET-COVER to an instance (I, J) (a LAV$^{=,\neq}$-schema mapping) of OPTIMAL-REPAIR$_{\text{LAV}^{=,\neq}}(\mathbf{S}, \mathbf{T})$.

We must show that the following hold for suitable $\alpha, \beta > 0$.

(a) For all instance Ins of MIN SET-COVER, algorithm f produces an instance $(I, J) = f(Ins)$ such that $OPT((I, J)) \leq \alpha \cdot OPT(Ins)$, where $OPT((I, J))$ is the cost of the optimal LAV schema mapping for (I, J) in the LAV$^{=,\neq}$ repair language, and

$OPT(Ins)$ is the cost of an optimal solution for the Min Set-Cover instance Ins.

(b) For all vfe LAV$^{=,\neq}$-schema mappings for (I, J) with size c', there is a polynomial-time algorithm g that produces a set cover for Ins with cost c such that $|c - OPT(Ins)| \leq \beta|c' - OPT(Ins')|$.

Since we know that $OPT(Ins) = k$ iff $OPT((I, J)) = 11k$, property (a) holds for $\alpha = 11$.

For property (b), suppose that we are given an arbitrary vfe LAV$^{=,\neq}$-schema mapping $\mathcal{M} = \{\mathbf{S}, \mathbf{T}, \Sigma\}$. By the arguments given in the proof of Theorem 2.23, we may assume without loss of generality that Σ consists entirely of LAV$^{=,\neq}$-constraints of the form

$$t_i = P(x, y_1, y_2, y_3, y_4) \wedge (x = i) \rightarrow Q(y_1, y_2, y_3, y_4)$$

Note that the relevant arguments in the proof of Theorem 2.23 involve polynomial-time computable transformations. In particular, we are using the fact that the problem whether a LAV$^{=,\neq}$-constraint explains a target fact is polynomial-time computable. The fact is true because we can chase the source instance with the given LAV$^{=,\neq}$-constraint and see if the chase result contains the target fact in question. Note that the chase procedure, introduced in [30], runs in polynomial time.

We define the function $g(\mathcal{M})$ to be $\{\ i \mid t_i \in \Sigma\ \}$, and $g(\mathcal{M})$ is a set cover for Ins. Let c be the cardinality of $g(\mathcal{M})$. We show that $g(\mathcal{M})$ satisfies the property (b). Let c' be the size of $\mathcal{M}$. Obviously, c' is a multiple of 11 because each constraint of the form t_i in $\mathcal{M}$ has cost 11. Moreover, we have that $c' \geq 11c$ because the number of constraints in $\mathcal{M}$ is at least c. It follows that we have following inequality:

$$|c - k| = \frac{1}{11}|11c - 11k| \leq \frac{1}{11}|c' - 11k|$$

This shows that property (b) holds with $\beta = \frac{1}{11}$. The theorem is proved. Note that the proof for the $\mathcal{L}^{=}$ case follows immediately from the proof of $\mathcal{L}^{=,\neq}$ case. □

It was shown in [60] that there is a constant c such that there is no polynomial time $c\ln(n)$-approximation algorithm for MINIMUM SET COVER, where n is the size of the universe of a given set-cover problem instance. The L-reduction established in Theorem 2.34 translates every MINIMUM SET COVER problem instance with universe of size n into an OPTIMAL-REPAIR problem instance consisting of a single data example with at most n target facts. Therefore, Theorem 2.33 is proved.

2.5 Experimental Evaluation

We implemented the approximation algorithm presented in the previous section. In this section, we present an experimental evaluation of the proposed $\mathcal{H}(n)$-approximation on a real-world schema mapping scenario.

2.5.1 Optimized Implementation

The first step of the approximation algorithm, described in Figure 2.1, consists of two phases:

(1) *Constraints-generation phase*: compute all SH-LAV$^{=}$-constraints for the input set E of data examples.

(2) *Validity-checking phase*: check the validity of each constraint obtained in Step 1 w.r.t. E, and omit the constraints that are invalid.

The Constraints-generation phase (Phase 1) is computationally expensive, and must be optimized so that the approximation algorithm is practically feasible. We have applied several optimizations to improve the efficiency of the implementation. The optimizations help reduce the space of SH-LAV$^{=}$-constraints that is considered by the algorithm. In this way, the optimized algorithm can produce vfe SH-LAV$^{=}$-schema mappings that are still within the $\log(n)$-factor approximation ratio. In our implementation, we have applied two kinds of optimizations: (1) *zero-amplification* optimizations, and (2) *constant-amplification* optimizations.

Intuitively, zero-amplification optimizations prune away constraints that would not be picked by the greedy algorithm. Hence, the approximation bound remains unchanged by this kind of optimizations. For instance, one of the zero-amplification optimizations we applied is the *Global Threshold Rule.* This rule is established based on the observation that, for a given ground data example (I, J), the schema mapping consisting of all facts in J is a trivial vfe schema mapping for (I, J). We can therefore use the cost of the trivial vfe mapping as a global threshold so that we can ignore every candidate constraint whose size is greater than or equal to this threshold. Clearly, such an optimization never increases the $\log(n)$-approximation bound. In contrast, we also consider constant-amplification optimizations which may improve the running time of the algorithm but at the expense of increasing the approximation bound by a fixed constant-factor. Example 2.35 below illustrates both the algorithm and some of the optimizations we have implemented.

Example 2.35 In this example, we will first illustrate the workflow of the approxi-amtion algorithm (PART I) and then illustrate an example of constant-amplification optimizations (PART II).

PART I- workflow of the approximation algorithm.

Consider the data example (I, J) shown in Table 2.1. There is no vfe SH-LAV schema mapping for (I, J) and that there are only five pairwise logically inequivalent SH-LAV constraints (up to variable renaming) for these schemas:

σ_1:Geo$(x, x) \rightarrow$City(x) σ_2:Geo$(x, y) \rightarrow$City(y) σ_3:Geo$(x, y) \rightarrow$City(x)

σ_4:Geo$(x, x) \rightarrow \exists m$City$(m)$ σ_5:Geo$(x, y) \rightarrow \exists m$City$(m)$

Each of the five SH-LAV constraints has many possible SH-LAV$^=$ repairs. The first step of the algorithm computes the set of all SH-LAV$^=$ constraints (up to renaming of variables) that are valid in (I, J), and all ground target facts in J. Our implementation will perform the following optimizations to avoid searching through the large number of SH-LAV$^=$-constraints in general.

For our example, observe that all SH-LAV$^=$-repairs of σ_1 can be immediately disregarded because their left-hand side cannot be satisfied in I. The same argument applies to σ_4. Similarly, all SH-LAV$^=$-repairs of the SH-LAV constraint σ_5 can be omitted, because the only way they could explain a target fact is if they have a conditional equality in their right-hand side of the form $(x = c \rightarrow m = d)$ or $(m = d)$. Each implication of the form $(x = c \rightarrow m = d)$ costs 4 and only helps explain one target fact. The equality of the form $(m = d)$ costs 2 but the entire constraint can only explain one target fact. Hence, the cost would be lower if we had simply included the corresponding target facts as ground facts in the SH-LAV$^=$ schema mapping. These optimizations are examples of zero-amplification optimizations, because they do not increase the approximation bound. Next, consider σ_3. We know that there is no SH-LAV$^=$-repair of σ_3 that is both valid for (I, J) and that explains even a single fact (in general, we can restrict attention to $\mathcal{L}$-constraints where a universally quantified variable occurs in a specific position of the right-hand side, only when one of the data examples actually contains facts matching it). This is another example of a zero-amplification optimization.

To summarize, for this data example, it is sufficient to consider SH-LAV$^=$ repairs of σ_2, and ground target facts, without affecting the quality of the schema mapping produced by the algorithm. Finally, observe that it is sufficient to consider only equalities that "are realized in the data example" when considering repairs of σ_2. In other words, it is unnecessary to consider a repair of σ_2 that includes equalities such as $(x = \text{US}) \wedge (y = \text{London})$ because no fact of I will satisfy these equalities.

After applying these optimizations, the number of candidate SH-LAV$^=$-constraints is 57 (plus ground target facts).

Next, the algorithm will repeatedly pick a constraint or ground fact that has highest cost effectiveness at each iteration until all facts from J are explained. The algorithm will perform the following in order

- 1: select "Geo(x,y)$\wedge$(x=US)$\rightarrow$ City(y)", which explains 5 previously-unexplained facts

- 2: select "Geo(x,y)$\wedge$(x=California)$\rightarrow$City(y)", which explains 2 previously-unexplained facts
- 3: add a ground fact "City(Toronto)".

After these three iterations, the algorithm terminates and output the following SH-LAV schema mapping:

$$\mathcal{M}_{greedy}=\{\text{Geo(x,y)} \wedge \text{(x=US)} \rightarrow \text{City(y)},\ \ \text{City(Toronto)}$$
$$\text{Geo(x,y)} \wedge \text{(x = California)} \rightarrow \text{City(y)}\}$$

Hence, our algorithm obtains a SH-LAV schema mapping of $size(\mathcal{M}_{greedy}) = 13$. The following SH-LAV$^{=}$-schema mapping (which can be shown to be minimal) has size 10.

$$\{\text{Geo(x,y)}\wedge\text{(x=Calif)}\rightarrow \text{City(y)},\ \ \text{Geo(x,y)}\wedge\text{(x=NorthAm)}\rightarrow \text{City(y)}\}$$

PART II - constant-amplification optimizations.

The first step of our approximation algorithm is to compute a set of candidate SH-LAV-constraints (in the base language) together with their SH-LAV$^{=}$-repairs which will be later used by the Set-Cover greedy algorithm. To illustrate the constant-amplification optimizations, consider the following simple data example (I, J) and a SH-LAV-constraint t for which we need to compute repairs.

$$I = \{S(a, 1), S(b, 2), S(c, 3)\}, \quad J = \{T(a, a, a), T(b, b, b)\}$$

$$t : S(x, y) \rightarrow \exists m\ T(m, m, m)$$

The optimal SH-LAV$^=$-repair of t that is vfe for (I, J) is

$$t' : S(x,y) \rightarrow \exists m\ T(m,m,m) \wedge (y = 1 \rightarrow m = a) \wedge (y = 2 \rightarrow m = b)$$

The constraint t' uses multiple conditional equalities to explain more than one target fact. In practice, computing all possible combinations of conditional equalities that are valid and explaining for multiple target facts is computationally expensive. In order to make the runtime be within reasonable range, our implementation adds only one conditional equality to each existentially quantified variable in a constraint in consideration. If we want to add an additional conditional equality over the same existentially quantified variable, we make a copy of the original SH-LAV-constraint and then add the new conditional equality. For instance, for the data example (I, J) and t shown above, our implementation will generate two SH-LAV$^=$-constraints as follows:

$$t_1 : S(x,y) \rightarrow \exists m\ T(m,m,m) \wedge (y = 1 \rightarrow m = a)$$

$$t_2 : S(x,y) \rightarrow \exists m\ T(m,m,m) \wedge (y = 2 \rightarrow m = b)$$

Clearly, t' and $\{t_1, t_2\}$ are both vfe for (I, J), but $\{t_1, t_2\}$ has larger size. We have results that show that this type of optimizations may increase the size of a schema mapping but the increase in the size of the schema mappings is bounded by a constant-factor that depends only on the source and target schemas (assuming schemas are fixed). Altogether, this means that we are able to improve the runtime efficiency at the expense of increasing the cost of the solution by a fixed constant factor. $\square$

2.5.2 Experiments

We now demonstrate that the optimized approximation algorithm is practically feasible in a real-world mapping scenario, and show the quality of the mappings returned by the algorithm is good.

Mapping Scenario

We use the mapping scenario derived from the OBDA (Ontology-Based Data Access) mappings in [50]. As part of the Ontop project, the authors of [50] developed an ontology for movies [2] and map the schema of the IMDB database to the ontology so as to populate the movie ontology with data from the IMDB database. The Ontop project contains a mapping consisting of a collection of OBDA constraints, and that were developed by UNIBZ students as part as a lab assignment and later improved by the research team. We use their OBDA constraints as the "ground truth" and compare the constraints returned by our approximation algorithm against these constraints to understand how well our algorithm performs.

Statistics of source and target schemas

The movie ontology uses a $\langle subject, predicate, object\rangle$-model. In our experiment, we turn every $\langle subject, predicate, object\rangle$ ontological concept into a binary fact $predicate(subjuect, object)$. We draw data examples from IMDb dataset (source) and Movie ontology (target). IMDB dataset contains twenty-one relations and eight of them

[2]see `https://github.com/ontop/ontop/wiki/Example_MovieOntology` for more details

are used in the mapping from IMDB to movie ontology (A full description of the IMDB schema can be found at [51], and the OBDA mappings that are constructed by hand can be found at [49]). There are sixty-eight authored OBDA constraints and therefore there are sixty-eight movie ontology concepts (i.e., sixty-eight target binary relations). A close examination reveals that these sixty-eight OBDA constraints can be categorized into following four types:

- **Type I**: Copying SH-LAV-constraints.

$$\text{E.g., name}(id, name) \rightarrow \text{HasBirthName}(id, name)$$

- **Type II**: Projection SH-LAV-constraints.

$$\text{E.g., company\_name}(id, name, country\_code, IMDB\_id, name\_nf, name\_sf) \\ \rightarrow \text{compHasName}(id, name)$$

- **Type III**: SH-LAV$^{=}$-constraints with one equality on the left-hand side.

$$\text{E.g., cast\_info}(id, person\_id, movie\_id, p\_role\_id, note, nr\_order, role\_id) \\ \wedge (role\_id = 8) \rightarrow \text{hasDirector}(movie\_id, person\_id)$$

- **Type IV**: SH-LAV$^{=}$-constraints with two equalities on the left-hand side.

$$\text{E.g., movie\_info}(id, movie\_id, info\_type\_id, info, note) \wedge (info\_type\_id = 3) \\ \wedge (info = Thriller) \rightarrow \text{SensibleThrilling}(movie\_id, info)$$

In our experiments, we choose some constraints in each category to test our algorithm. Concretely, we have chosen one copying constraint, one project constraint, and three constraints from Type III and Type IV (for Type III and IV, we choose those constraints whose source relations have different arities). We chose only one constraint from Type I and Type II because copying and projection are relatively simple transformations comparing with the transformations in Type III and Type IV. The statistics of selected IMDB relations and Movie ontology concepts are shown in Table 2.3.

Table 2.3: Statistics of source and target schemas

	IMDB	Movie ontology
number of relations	8	8
average arity	7.5	2
max. arity	11	2
min. arity	5	2

Adjusted mapping scenario

As mentioned, we use the OBDA constraints that were developed by human for the IMDB-to-Ontology mapping as ground truth. Hence, we can directly compare them with the constraints that are returned by our approximation algorithm. The IMDB database is populated with real data from IMDB.com and the size of the smallest table has about 219K records. Since our schema-mapping approximation algorithm is designed to run over small data examples and not over large datasets in general, we cannot simply execute our algorithm directly on the datasets obtained. Hence, in our experiments, we first generate small data examples from the real datasets before we apply our approximation algorithm and test the scalability of our algorithm by executing it on data examples of increasing sizes.

Data example generation

The purpose is to obtain a series of data examples of different sizes. We generate data examples with the following procedures. All OBDA constraints in the mapping specified in the Ontop project is of the form

$$S(\mathbf{x}) \wedge \alpha(\mathbf{x}) \rightarrow T(\mathbf{x})$$

where $\alpha(\mathbf{x})$ is of a collection of equalties $(x_1 = c_1) \wedge \cdots \wedge (x_n = c_n)$

We first take an OBDA constraint t, and then randomly sample k (e.g., $k = 10$) facts, denoted by $I = \{f_1, f_2, \ldots, f_k\}$, from the S-relation, and we chase the k sampled facts with t (see [29] for more details about the chase procedure for data exchange) to generate a set J of T-facts which will be served as target facts. It follows that, (I, J) will be the data example generated. To obtain larger data examples, we simply increase the sampling size and repeat the above procedure. For generating a set of m data examples (used in Experiment 4), we simply run the same procedures m times to obtain m data examples, and these examples form a set E of data examples.

Experiments

Our goal is to understand the actual running time of the implementation in various settings, and the quality of constraints returned by the approximation algorithm on the IMDB-to-Ontology scenario. The approximation algorithm was implemented in Java, with PostgreSQL v9.3 as the underlying database engine. All the experiments were running on an Intel Core i7-4770 3.4GHz CPU Linux machine with 16GB memory. Experiments presented here take a single ground data example as input unless otherwise stated. We are interested in the following aspects:

(a) Compare the running time of the approximation approach with the exhaustive algorithm (i.e., the exact algorithm).

(b) Scalability of the optimized implementation with respect to:

  – data examples of different sizes.
  – different number of source relations and target relations.

(c) Running time of each of the three phases of the approximation algorithm (i.e., constraints-generation phase, validity-checking phase, and greedy phase).

(d) Compare the constraints produced by our approximation algorithm with the ground truth OBDA constraints.

(e) Understand the runtime efficiency of our algorithm on multiple data examples.

Experiment 1 - Approximation algorithm vs. exhaustive algorithm. We show that the exhaustive algorithm is not feasible because it runs out of memory even for a small input data example. The exhaustive algorithm is implemented in a straightforward way: after computing the set C of candidate valid and explaining constraints for a given data example, the exhaustive algorithm will then compute all subsets of C and then try to find a a minimum subset of C that is vfe for the given data example. Obviously, this is a $O(2^n)$ algorithm where n is the size of C. We run our approximation algorithm and the exhaustive algorithm on three small data examples of sizes 10, 15, and 20 (total number of source and target facts). The results are list in Table 2.4.

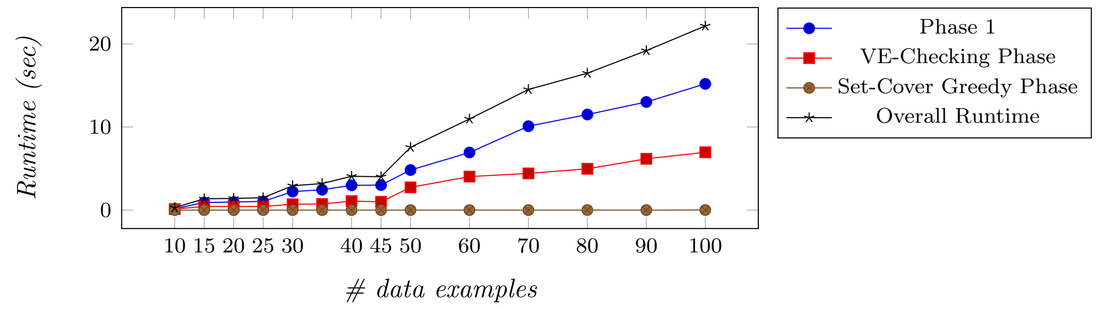
Table 2.4: Running time - approximation algorithm vs. exhaustive algorithm

Size of data examples	# candidate valid & explaining constraints	running time of approx. algorithm (seconds)	running time of exhaustive algorithm (seconds)
10	22	0.001	7.14
15	24	0.001	31.25
20	26	0.001	Out of memory

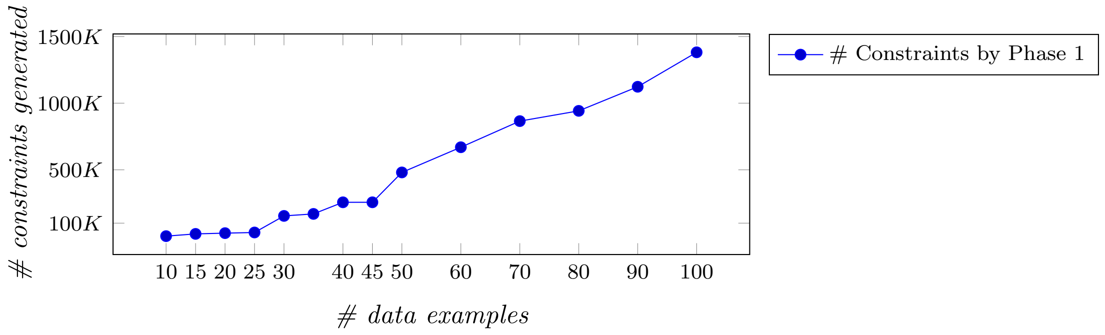
Clearly, as shown in Table 2.4, the exhaustive search performs much worse than the approximation algorithm in terms of running time. Table 2.4 shows that even for the case where the number of valid and explaining constraints is small, the exhaustive algorithm ran out of memory and therefore it is not feasible. On the other hand, the approximation algorithm is very efficient.

Experiment 2 - Scalability in terms of data examples of different sizes. In this experiment, the data examples used are composed of facts from two source relations and two target relations. The two source relations have arity five and seven respectively, and the two target relations are binary relations. These four relations are selected from two OBDA constraints in the ground truth solution (one from Type III and one from Type IV). We start our test with a small data example, and increase the sizes of data examples in the subsequent tests. Initially, the data example has six source facts from two source relations and four target facts from two target relations. In the last test, there are a total of 100 facts in the example tested (note that in each test, there are roughly 60% of the facts contained in the data examples are source facts, and roughly 40% are target facts). The overall running time is shown in Figure 2.2a. Moreover, the running time of each phase of our algorithm is also shown in the same figure.

Figure 2.2a shows that our algorithm executes fast for data examples with smaller number of facts (e.g., less than six seconds for a data example of size forty), and is still efficient for large data examples (e.g., finished within twenty-four seconds for a data example of size 100). The approximation algorithm is a high polynomial degree algorithm. The running time shows that the optimizations applied in the implementation is capable of improving the efficiency of the algorithm. Another observation is that the



(a) Overall running time with respect to data examples of different sizes



(b) Number of constraints generated in Phase 1

Figure 2.2: Runtime efficiency and size of generated constraints

greedy phase is extremely fast (in all tests, the greedy phase finished within a second). We also see that the constraints generation phase (i.e., Phase 1 of the algorithm) is the most time-consuming part of the algorithm. We did another experiment to record the number of constraints generated in Phase 1. The numbers are shown in Figure 2.2b. We can see that the runtime of Phase 1 and the number of constraints generated in Phase 1 are highly related. Moreover, the validity and explanation checking phase (i.e., VE-Checking phase) has similar runtime as Phase 1, this is because the runtime of VE-Checking phase is determined by the number of constraints generated in Phase 1.

Therefore, we can conclude that, Phase 1 contributes the most to the overall runtime of our approximation algorithm.

Quality of results in Experiment 2. We did fourteen tests in Experiment 2, and we compared the constraints returned by the approximation algorithm with the ground truth OBDA constraints. It turned out that our approximation algorithm produced optimal schema mappings in all cases.

Experiment 3 - Scalability in terms of different numbers of relations. In each test of this experiment, the data examples are constructed by drawing five facts from each source instance and five facts from each target instance, and we keep increasing the number of source relations and target relations. We increase the number of source relations and target relations in turn in each subsequent test. In this experiment, only constraints of Type III and Type IV are involved. Initially, there are one source relation and one target relation involved (total 10 facts), and eventually there are five source relations and five target relations involved (total 100 facts).

Table 2.5: Running time with respect to different number of relations

Total # of relations	# of source relations	# of target relations	max. source schema arity	# of constraints after validity checking	Running time of the approximation algorithm (seconds)
2	1	1	5	16	0.069
3	2	1	7	18	0.11
4	2	2	7	1908	0.275
5	3	2	7	3664	0.166
6	3	3	7	5414	0.382
7	4	3	7	6284	0.346
8	4	4	7	9667	0.356
9	5	4	11	14301	10.175
10	5	5	11	38465	48.284

The results are listed in Table 2.5. In the first seven tests, our algorithm executes to completion under one second in all cases. However, the runtime became significantly slow in the cases when larger (arity of eleven) source relations are involved. Concretely, in the last two tests, we included facts from a source relation of arity eleven. In contrast, in the first seven tests, all relations involved have arity either five or seven. The results in Table 2.5 are not surprising since our approximation algorithm has exponential complexity in the worst case when the schema is part of the input.

Quality of results in Experiment 3. We did nine tests in this experiment, and the approximation algorithm produced the same OBDA constraints that are specified in the ground truth in eight of the tests. There is one test in which our approximation algorithm produced a constraint that is different from the corresponding ground truth OBDA constraint. Concretely, the constraint t shown below is the one included in the ground truth, and the constraint t' is the one returned by our approximation algorithm. The constraint t' does not have the equality ($\texttt{info\_type\_id} = \texttt{3}$) showing on its left-hand side.

$$t : \text{movie\_info}(id, movie\_id, info\_type\_id, info, note) \wedge (info\_type\_id = 3)$$
$$\wedge\, (info = Thriller) \rightarrow \text{SensibleThrilling}(movie\_id, info)$$

$$t' : \text{movie\_info}(id, movie\_id, info\_type\_id, info, note) \wedge (info = Thriller)$$
$$\rightarrow \text{SensibleThrilling}(movie\_id, info)$$

We looked into the full IMDB dataset and verified that our approximation algorithm made the right decision. The equality ($\texttt{info\_type\_id} = \texttt{3}$) is redundant, since every tuple p in the `movie_info` table that has value "3" in attribute `info_type_id`, the tuple p also has value “`Thriller`” in attribute `info`. In other words, t and t' agree on validity and explanation, but t' has smaller size. In other words, our algorithm derived a constraint that is in fact better than the constraint that corresponds to our ground truth.

Experiment 4 - runtime efficiency on multiple data examples. In this experiment, we try to understand the runtime efficiency of our approximation algorithm on multiple data examples. In this experiment, the input to our approximation algorithm is a set E of data examples, and we test our algorithm with respect to different sizes of E (i.e., test with respect to different numbers of data examples in E). First, we selected four OBDA constraints (which are exactly those constraints shown in Section 2.5.2 as example constraints) to generate our data examples. We then randomly choose five facts from each source relation specified in the four OBDA constraints, and obtain target instance by chasing the randomly selected facts with the four OBDA constraints. To get $m \geq 2$ examples, we simply repeat the same procedure m times. We have tested our approximation algorithm on ten different sets of data examples, and the results are listed in Figure 2.3. We can see that the overall runtime grows linearly, and the runtime of Phase 1 has similar behaviour. However, the runtime of the validity and explanation checking phase grows a bit faster. Note that the data examples used in the experiments have comparable sizes due to the way we generate them. We run our approximation algorithm on each individual data example of the input set E of data examples and obtain a collection C of constraints. We then check the validity and explanation of C on every data example in E, and keep those that are valid and explaining for every data example in E. Hence, the runtime of VE-checking phase grows quadratically in the number of examples in the input set of data examples. On the other hand, the runtime of Phase 1 grows linearly in the size of input set of data examples because the algorithm generates constraints for each data example independently. However, as the number of examples grows, the runtime of our approximation algorithm will be dominated by the runtime of VE-checking phase, since it grows quadratically. Notice that, in this set of experiments, we no longer have the ground truth constraints since we now have multiple data examples as input (The ground truth OBDA constraints do not need to be vfe when there are multiple data examples). Therefore, we do not measure the quality of results in this experiment.

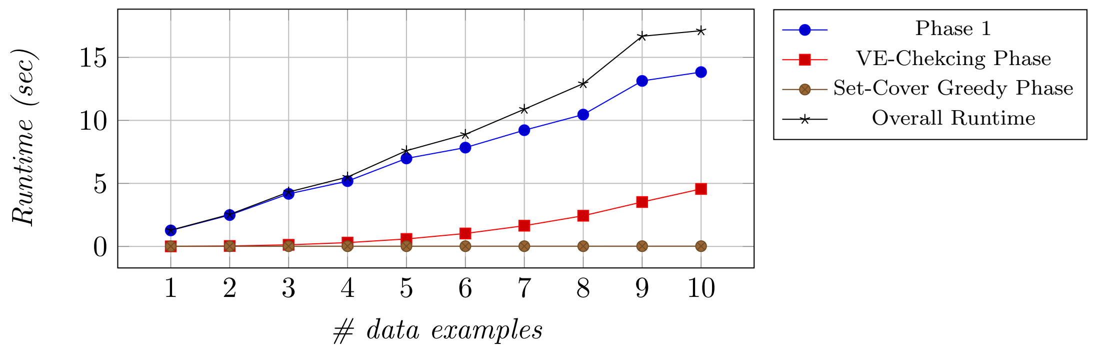


Figure 2.3: Runtime with respect to multiple data examples

Discussion. We have conducted experiments to evaluate our optimized implementation on a real mapping scenario. Our experimental results reveal that the optimized approximation algorithm is efficient for this mapping scenario. Moreover, the SH-LAV$^=$-schema constraints returned by the approximation algorithm for the case where the input consists of a single data example are as good as authored OBDA constraints which we take as ground truths. In fact, our approximation algorithm produced a constraint that is better than the authored OBDA constraint in one of the data examples. We also validated that our approximation algorithm is efficient when the input consists of multiple data examples. Our experimental results show that the performance of the algorithm is dominated by the arity of the schemas; as the schemas are larger, our algorithm runs slower. In fact, if schemas are part of the input, our approximation algorithm runs in exponential time, and that is inevitable.

2.6 Conclusions

We investigated the derivation of an optimal schema mapping from a set of data examples and focused on the approximation properties of this optimization problem. We considered several different sublanguages of the language of GLAV schema mappings and delineated the boundary as regards good approximation properties. To this effect, we established negative results about the existence of approximation algorithms for GAV and GLAV schema mappings, but also showed that the collection of SH-LAV schema mappings is the largest subclass of GLAV mappings for which we can obtain a positive approximation result. These results pave the way for leveraging further the rich area of approximation algorithms and applying it to schema-mapping discovery.

We enchanced our approximation algorithm with heuristic rules, implemented it, and evaluated it on a real-world schema mapping scenario. Our experimental evaluation suggests that this approach may indeed be applied in practice to construct a near-optimal schema mapping from a set of data examples.

An important question that remains to be answered is the existence of a polynomial time $\mathcal{H}(n)$-approximation algorithms for computing near-optimal LAV$^{=,\neq}$ schema mappings, as well as near optimal SH-LAV$^{=,\neq}$ schema mappings.

Chapter 3

Learning GAV Schema Mappings from Universal Examples

In this chapter, we introduce a PAC learning algorithm for GAV schema mappings. The work presented in this chapter is built on the learning framework introduced in [21]. Our work is motivated by the following scenario. Suppose that the user is provided with a black box implementing some schema mapping $\mathcal{M}$, whose specification is unknown due to proprietary reasons. By using the provided black box, we can convert a source instance I to a target instance J such that the underlying mapping $\mathcal{M}$ is satisfied. A natural question to ask is: *"is there a way to recover the specification of $\mathcal{M}$ from the black box?"*. We can attempt to answer this question if we can collect a set of data examples. The derivation of the specification of a schema mapping from data examples can be seen as a reverse engineering problem, that is, a problem in which the specification of a "goal" device has to be discovered from the behavior of that device. In our problem, the mapping $\mathcal{M}$ hidden in the black box is the goal device that we are trying to reveal. Data examples that satisfy $\mathcal{M}$, in a precise sense, describe the data-exchange behavior of the goal schema mapping. Going back to our motivating scenario, in practice, users may have some pre-existing source instances at hand, or, a set of source instances can be collected from some data source. In either case, users can collect a set of data examples (using the provided black box). Then the main question to answer is: *"is there a way to reveal the specification of the underlying schema mapping using both the collected data examples and the provided black box?"*.

At first glance, the three major example-driven schema-mapping discovery approaches (i.e., the fitting, the Gottlob-Senellart, and the learning approaches) all look adequate for this problem. However, there are essential differences among them. By a careful inspection, we notice that the fitting framework and the Gottlob-Senellart framework both take into account only the set of data examples[1]. In other words, the two frameworks do not take into account the goal schema mapping even if it is provided as an oracle. As a result, the two frameworks would just produce a schema mapping that is fitting (or, valid and fully explaining) for the input set of data examples. Concretely, the fitting framework would, given a set E of data example, return the canonical GLAV schema mapping which is obtained by spelling out the facts of E (assuming there is a fitting GLAV mapping for E). By construction, the canonical GLAV mapping of E is fitting for E [3]. As already discussed in Section 1.1, such a canonical GLAV schema mapping will unlikely work well on future data examples because it is very specific

[1]Note that the work presented in Chapter 2 extends the Gottlob-Senellart framework by allowing multiple data examples.

to the input set of data examples. The Gottlob-Senellart framework proposed a cost model that would avoid producing a mapping that is a full description of the input. Recall that, given a set E of ground data examples, the cost model of Gottlob-Senellart framework would produce a valid and fully explaining mapping for E with minimum size (syntactically). One shortcoming of the cost model is that, there could be multiple logically inequivalent schema mappings that achieve the same optimality for the same input. Since the cost model does not take into account the goal schema mapping, the Gottlob-Senellart approach would arbitrarily choose one of the optimal schema mappings. Even if there is only one optimal schema mapping, the optimal mapping may not be the "right" mapping.

As opposed to the fitting and the Gottlob-Senellart frameworks that make use of only the input set of data examples, the learning framework provides a way to learn schema mappings by using both the input set of data examples and the goal schema mapping, which is provided as an oracle. The learning framework views schema mappings as concepts that can be identified by data examples (e.g., positive examples and negative examples). The task is then to identify the goal concept (i.e., goal schema mapping) by asking a number of queries about it to different oracles. Typical queries include: *equivalence queries* (whether two schema mappings are logically equivalent?); *labeling queries*[2] (return the intended target instance, e.g., the universal solution, of an input source instance w.r.t. the goal mapping). Our motivating scenario can be naturally viewed as a learning problem, where the provided black box is both the goal

[2]In the learning framework, membership queries were considered. We will show shortly that membership queries are special cases of labeling queries.

schema mapping that we are trying to learn and an oracle that answers labeling queries.

In [21] the learnability of GAV schema mappings with respect to different learning models (including Angluin's exact learning model [5] and Valiant's PAC model [69]) was studied. In particular, the author presented a concrete learning algorithm, which we call EXACTGAV, that efficiently and exactly learns GAV schema mappings using both a labeling oracle and an equivalence oracle. In addition to an exact learnability of GAV schema mappings, by a series of reductions shown in [21], the authors showed that there exist a PAC learning algorithm, which makes use of a labeling oralce, for GAV schema mappings. However, no concrete PAC learning algorithm was spelled out in [21]. The goal of our work is to develop a practical learning algorithm that, given a goal GAV mapping $\mathcal{G}$ (available as an oracle) and given a set of universal examples for $\mathcal{G}$, produces a GAV mapping that is semantically close to $\mathcal{G}$. In this chapter, we present a learning algorithm for GAV schema mappings. We show that the proposed algorithm is an Occam algorithm [15] leading up to a PAC learning algorithm for GAV schema mappings. We implemented the aforementioned algorithm and carried out an experimental evaluation using mapping scenarios created by a state-of-the-art benchmarking tool called iBench [8].

The rest of the chapter is organized as follows: In Section 3.1, we introduce concepts and notations needed and related prior work. In Section 3.2, we present the promised PAC learning algorithm for GAV schema mappings. In Section 3.3, we present an experimental evaluation of the proposed learning algorithm. Finally, in Section 3.4, we summarize our results and discuss open questions.

3.1 Preliminaries and Prior Work

In this section, we introduce the concepts and notations relating to computational learning theory. Note that the preliminaries introduced in Section 2.1 are also used in this chapter.

Computational learning models. We introduce two major computational learning models:

(1) the *exact learning model* introduced by Angluin [5];
(2) the *PAC (probably-approximately-correct) model* introduced by Valiant [69].

Before defining the two learning models, we first introduce the following notations.

Labeled examples and concept. Let X be a (possibly infinite) set of examples. A *labeled* example is an element of $X \times L$, where L is a set of labels. In most machine learning scenarios, $L = \{0, 1\}$ (i.e., positive and negative).

A *concept over* X is a function $c : X \to \ell$ where $\ell \in L$, and *a concept class* $\mathcal{C}$ is a collection of concepts. We assume that concepts are specified by some representation system, that is, a *representation system for a concept class* $\mathcal{C}$ is a string language $\mathcal{L}$ over some finite alphabet, together with a surjective function $r : \mathcal{L} \to \mathcal{C}$. For every concept representation $l \in \mathcal{L}$, we write $|l|$ to denote its length. As a concrete example, suppose that $X = \{0, 1\}^n$ is the set of all n-ary boolean assignments. Then we could consider the concept class consisting of all Boolean functions $c : \{0, 1\}^n \to \{0, 1\}$ specified by Boolean formulas in DNF (disjunctive normal form) formulas over the set of variables $x_1, \ldots, x_n$.

Labeling and equivalence queries. We often assume that the learning algorithm is provided with various oracles (depending on the learning task at hand) that have access to the goal concept such that these oracles can answer specific queries about it. For every concept c, we denote by LABEL_c the *labeling oracle for* c, that is, the oracle that takes as input an example $x \in X$ and return its label, $c(x)$ (i.e., some $\ell \in L$), according to c. Note that when $L = \{0, 1\}$, labeling oracles are also known as *membership* oracle.

For every concept c, we denote by EQ_c the *equivalence oracle for* c, that is, the oracle that takes as input another concept c' and return "Yes" if c and c' are logically equivalent (i.e., $c \equiv c'$); otherwise, it returns a counterexample x (i.e., an example x such that $c(x) \neq c'(x)$).

Exact learning. We say an algorithm `alg` *exactly learns a concept class* $\mathcal{C}$ *with labeling and/or equivalence queries* if, for all natural number n and for all concept $c \in \mathcal{C}$ with $size(c) \leq n$, when `alg` is run on input n and with a labeling oracle LABEL_c and/or an equivalence oracle EQ_c, it outputs a representation of c. A concept class $\mathcal{C}$ is *efficiently exactly learnable with labeling and/or equivalence queries* if there is an exact learning algorithm with labeling and/or equivalence queries for $\mathcal{C}$ that runs in polynomial time.

PAC learning. The task of a PAC learning algorithm is to approximately learn an unknown goal concept $c \in \mathcal{C}$ based on the labeled examples that have been randomly generated according to some probability distribution. Recall that $[0, 1]$ is the closed unit interval of the real numbers, and recall that a *a probability distribution* over X is a function $D : X \to [0, 1]$ such that $\Sigma_{x \in X} D(x) = 1$. For every concept $c \in \mathcal{C}$ and probability distribution $D : X \to [0, 1]$, we denote by $\mathrm{EX}_{c,D}$ the *example oracle* for c

and D, that is, the oracle that, upon request, returns a labeled example $(x, c(x))$, where $x \in X$ is randomly chosen according to the probability distribution D.

A PAC learning method takes as input: a rational accuracy parameter $0 < \epsilon < 1$, a rational confidence parameter $0 < \delta < 1$, and a natural number n, and that has access to an example oracle for an unknown goal concept of size at most n with respect to a probability distribution D. The PAC learning method must halt and return a candidate concept h. In particular, the returned concept h needs not to be exactly the same as the goal concept, instead, it is rather an approximation of it. We then can define the *error*, denoted by $\texttt{ERR}(h)$, for the returned concept h with respect to D and c as

$$\texttt{ERR}_{c,D}(h) = \mathop{\mathbf{Pr}}_{x \in D}(c(x) \neq h(x))$$

where $\mathbf{Pr}_{x \in D}(c(x) \neq h(x))$ denotes the probability that the event $c(x) \neq h(x)$ when $x \in X$ is drawn from the probability distribution D.

We say that an algorithm `alg` *PAC learns a concept class* $\mathcal{C}$ if for all rational number ϵ and δ with $0 < \epsilon, \delta < 1$, for all natural numbers n, for all concepts c with $size(c) \leq n$, and for all probability distributions D over the example set X of $\mathcal{C}$, when `alg` is run with inputs ϵ, δ, n and oracle $\mathrm{EX}_{c,D}$, the algorithm `alg` outputs with probability at least $1 - \delta$ a concept h with $\texttt{ERR}(h) \leq \epsilon$. A concept class $\mathcal{C}$ is *PAC learnable* if there is a PAC algorithm `alg` that learns it. If `alg` also runs in polynomial time then we say that $\mathcal{C}$ is *efficiently PAC learnable.* If a concept class can be learned by a (polynomial time) PAC algorithm with access to labeling oracle, then the concept class $\mathcal{C}$ is said to be *(efficiently) PAC learnable with labeling queries.*

Occam learning and its relation to PAC learning. Occam learning is another important model of algorithmic learning [15]. Recall the definition of an Occam algorithm.

Definition 3.1 [15] An Occam algorithm for a concept class C with constant parameter $0 \leq \alpha < 1$ and $k \geq 1$ is an algorithm A that takes as input a collection of $(x_1, c(x_1)), \ldots, (x_m, c(x_m))$ of examples labeled according to some unknown concept $c \in C$ and produces a hypothesis h consistent with the input of size at most $m^\alpha n^k$ where n is the size of c.

Note that, unlike [15], the above definition does not require the algorithm A to run in time polynomial in the combined size of the input examples. Blumer et al. [15] proved the following theorem.

Theorem 3.2 [15] If there is an efficient Occam algorithm for some concept class C, then C is efficiently PAC learnable.

In particular, if A is an efficient Occam algorithm for C, then we can obtain an efficient PAC learning algorithm A' for C by (1) asking for m many random examples with

$$m = \left(\frac{n^k \ln 2 + \ln (2/\delta)}{\epsilon} \right)^{1/(1-\alpha)}$$

that are fed to A, and (2) output the hypothesis that is returned by A. It follows that A' is an efficient PAC learning algorithm for C.

GAV schema mappings as concepts. Fix a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a GAV schema mapping (where Σ is a finite set of GAV constraints over $\mathbf{S}$ and $\mathbf{T}$). In [21] , the authors showed that we can identify a GAV schema mapping by all its universal examples, where the example space is the set of all source instances and the corresponding canonical universal solutions are labels. A GAV schema mapping $\mathcal{M}$ can be seen as a function that maps every source instance I into its canonical universal solution $\texttt{can-sol}_{\mathcal{M}}(I)$ (i.e., the label of I according to $\mathcal{M}$). We shall denote by GAV($\mathbf{S}$,$\mathbf{T}$) the concept thus defined.

The following are relevant results from [21]. We remark that the following results are not exactly stated as they were in [21]. In particular, the following results are formulated in terms of universal examples, whereas the corresponding results in [21] were formulated in terms of positive/negative examples. In [21], the authors showed that there is no fundamental difference between learning from positive/negative examples and learning from universal examples. In this chapter, we consider universal examples, and thus we formulate the following results from [21] in terms of universal examples.

Theorem 3.3 [21] Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ a target schema. If $\mathbf{S}$ contains a relation symbol of arity at least two, then GAV($\mathbf{S}$,$\mathbf{T}$) is not efficiently exactly learnable with labeling queries.

Theorem 3.3 shows that GAV($\mathbf{S}$,$\mathbf{T}$), in general, is not efficiently and exactly learnable if only a labeling oracle is provided. This negative result indicates that there is no efficient and exact learning algorithm for our motivating problem. However, ten Cate et al. showed that, if both a labeling oracle and an equivalence oracle are available then GAV($\mathbf{S}$,$\mathbf{T}$) is efficiently and exactly learnable (see the following theorem from [21]).

Theorem 3.4 [21] For every source schema $\mathbf{S}$ and every target schema $\mathbf{T}$, the concept class of all GAV mappings from $\mathbf{S}$ to $\mathbf{T}$ is efficiently exactly learnable with equivalence and labeling queries.

In addition to Theorem 3.4, ten Cate et al. presented a concrete algorithm (i.e., the EXACTGAV algorithm) that efficiently and exactly learns GAV schema mappings using both labeling queries and equivalence queries. However, the EXACTGAV algorithm is not practical because, unless the required equivalence oracle is provided, it is impossible to implement the equivalence oracle in question. Concretely, in order to implement the equivalence oracle, one would have to test, whether a candidate mapping $\mathcal{H}$ and a goal mapping $\mathcal{G}$ (given as an oracle) produce the same solution for every possible input source instance. Naturally, one may consider learn GAV schema mappings approximately. By a series of reductions shown in [21], the following corollary was presented.

Corollary 3.5 [21] For every source schema $\mathbf{S}$ and every target schema $\mathbf{T}$, the concept class of all GAV mappings from $\mathbf{S}$ to $\mathbf{T}$ is efficiently PAC learnable with labeling queries and an oracle for NP.

Although Corollary 3.5 shows the existence of a PAC learning algorithm for GAV($\mathbf{S}$,$\mathbf{T}$) using labeling queries, no concrete algorithm was spelled out in [21]. In this chapter, we propose a PAC learning algorithm, which we call GAVLearn, for GAV($\mathbf{S}$,$\mathbf{T}$). The GAVLearn algorithm was adapted from the EXACTGAV algorithm presented in [21]. It turned out that GAVLearn is in fact an Occam algorithm leading up to a PAC learning algorithm for GAV($\mathbf{S}$,$\mathbf{T}$). Moreover, if GAVLearn is provided with an oracle for NP then

GAVLearn is an efficient PAC learning algorithm for GAV($\mathbf{S}$,$\mathbf{T}$).

A Connection to Learning Union of Conjunctive Queries. GAV schema mappings, in a precise sense, are equivalent to union of conjunctive queries. Concretely, given a source instance I, chasing a GAV constraint t with I to obtain a target instance J can be seen as, using the left-hand side of t as a conjunctive query and execute it on I, and the the instance J is the query result. It follows that a GAV schema mapping, which consists of multiple GAV constraints, corresponds to a union of conjunction queries. Therefore, learning GAV schema mappings is equivalent to learning union of conjunctive queries. Conjunctive queries are important because they are supported by every relational database system. Futhermore, conjunctive queries correspond to the SELECT-FROM-WHERE queries in SQL, where the WHERE clause contains only equalities.

There have been a number of papers working on learning conjunctive queries. Active learning approaches based on human interactions have been proposed in [17,18,52]. In these approaches, queries are learned from raw data, and the learning process involves a human user who labels "informative" tuples selected by the algorithm. The assumption of these works was different from ours. They assumed that the user has a query in mind. However, we assume that the user has no prior knowledge of the goal GAV schema mapping, but an oracle access to the goal mapping is provided. In practice, the goal schema mapping we are trying to learn may contains tens of complex GAV constraints, which is extremely difficult for a human user to maintain. Moreover, they assumed that their algorithm is always able to produce the goal query from human interactions. Therefore, the focus of these approaches is not to learn a query that is logically equivalent to the goal query but rather to minimize the human effort needed.

There is another line of research that focuses on learning queries based on the knowledge of database integrity constraints [64, 68, 73]. One of the key ingredients of the approaches proposed in these works is a concept called *schema graph*, which is based on database integrity constraints, such as primary key and foreign key constraints. Among the three, [68] (which was an extended version of [67]) proposed a system called TALOS that supports reverse engineering union of conjunctive queries from a given data example. In Section 3.3, we present an comparison study of GAVLearn and TALOS with respect to synthetically generated mapping scenarios.

3.2 The GAVLearn Algorithm

In this section, we first introduce several lemmas and concepts leading up to the GAVLearn algorithm for GAV($\mathbf{S}$,$\mathbf{T}$). We then present the GAVLearn algorithm and prove that it is an Occam algorithm. We also provide a concrete example to illustrate the workflow of the GAVLearn algorithm.

Before we present the GAVLearn algorithm, we need several concepts and lemmas, some of which were introduced in [21]. The basic idea of the GAVLearn algorithm is to iteratively learn and maintain a GAV schema mapping $\mathcal{H}$ that consists of constraints that are *critically sound* with respect to the goal mapping $\mathcal{G}$, and it terminates when $\mathcal{H}$ fits E.

We say a GAV constraint C is critically sound with respect to $\mathcal{G}$ if (1) $\mathcal{G}$ logically implies C, that is, for every example (I, J) that satisfies $\mathcal{G}$ we have that (I, J) also satisfies C, and (2) for every GAV constraint C' obtained by removing one of the conjuncts of the left-hand side of C, we have that $\mathcal{G}$ does not logically imply C'.

Same as [21], we will also often identify a GAV constraint $\forall \mathbf{x}(\phi \rightarrow \psi)$ with the pair (I_ϕ, J_ψ) where I_ϕ, J_ψ are the canonical instances of ϕ and ψ (note that J_ψ must consist of a single fact). We also identify the GAV constraint $\forall \mathbf{x}(\phi \rightarrow \psi)$ with the $(\mathbf{S} \cup \mathbf{T})$-instance $I_\phi \cup J_\psi$. In this way, a homomorphism $h : C \rightarrow C'$ between GAV constraints is a function that maps atomic formulas occurring in the left-hand side or right-hand side of C to atomic formulas occurring in the left-hand side or right-hand side of C' (we write $C \rightarrow C'$ to denote the existence of homomorphism from C to C').

The following lemma is presented in [21], and it is a direct generalization of a results by Chandra and Merlin [25].

Lemma 3.6 [21] For all GAV constraint $C = (I, F)$, the following are equivalent:

- $\mathcal{G}$ logically implies C;
- $C' \rightarrow C$ for some $C' \in \mathcal{G}$
- $F \in \texttt{can-sol}_{\mathcal{G}}(I)$.

The proof of Lemma 3.6 can be found in [21]. Here we remark that, in our setting, computing $\texttt{can-sol}_{\mathcal{G}}(I)$ is in polynomial time by using the chase procedure [29]. This further means that we have a simple way to check whether or not a GAV constraint $C = (I, F)$ is logically implied by a mapping $\mathcal{G}$ (we can simply verify if $\texttt{can-sol}_{\mathcal{G}}(I)$

contains F). We also have the following lemma.

Lemma 3.7 (A direct generalization of a lemma in [21]) Let $\mathcal{G}$ be the goal schema mapping in question. Given a GAV constraint $C = (I, F)$, if $\mathcal{G}$ logically implies C, then one can compute a GAV constraint $Crit_{\mathcal{G}}(C)$ in polynomial time with an access to a labeling oracle for $\mathcal{G}$, and $Crit_{\mathcal{G}}(C)$ has following properties.

(1) $Crit_{\mathcal{G}}(C) \subseteq C$ ($Crit_{\mathcal{G}}(C)$ and C are viewed as instances)
(2) $Crit_{\mathcal{G}}(C)$ is critically sound with respect to $\mathcal{G}$.

Proof. To compute $Crit_{\mathcal{G}}(C)$ we can start with I and then try to repeatedly remove facts of I, as long as $\texttt{can-sol}_{\mathcal{G}}(I')$ contains F, where I' is the subinstance obtained from I. We stop the procedure until a minimal subinstance I' of I is reached. The source instance $I' \subseteq I$ obtained in this way is, by construction, such that the constraint (I', F) is critically sound with respect to $\mathcal{G}$. Note that there are $|I|$ many facts, and therefore at most $|I|$ many iterations are needed. □

For two GAV constraints C, C' that contain the same target relation, we denote by $C \times C'$ the GAV constraint that, viewed as an instance, the *direct product* of C and C', if well defined. Note that, in general, $C \times C'$ may not be well defined even if C and C' have the same target relation. However, in what follows, whenever we take a product of two GAV constraints (in the GAVLearn algorithm), the result is well defined. In fact we have the following lemma (introduced in [21]).

Lemma 3.8 [21] Let C, C_1, C_2 be GAV constraints, such that there are homomomorphisms $h_1 : C \to C_1$ and $h_2 : C \to C_2$. Then $C_1 \times C_2$ is well defined.

```
Input: $\mathcal{G}$ - an oracle to the goal mapping; $E$ - a set of universal examples for $\mathcal{G}$
Output: a mapping that fits $E$.
 1: $\mathcal{H} \leftarrow \emptyset$
 2: while true do
 3:     if for every example $(I, J) \in E$,
               it holds that $J = \texttt{can-sol}_{\mathcal{H}}(I)$ then
 4:         return $\mathcal{H}$
 5:     end if
 6:     choose an $(I, J) \in E$ such that $J \neq \texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$
 7:     /* in the proof of Theorem 3.9, we will show that
        $\texttt{can-sol}_{\mathcal{H}}(I)$ is a solution for $I$ w.r.t. $\mathcal{H}$ but not
        a solution for $I$ w.r.t. $\mathcal{G}$, and $\texttt{can-sol}_{\mathcal{H}}(I) \subsetneq J$ */
 8:     $F \leftarrow$ choose a fact $F \in J \setminus \texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$
 9:     Let $C$ be the canonical GAV constraint of $(I, F)$
10:     if $\mathcal{G}$ logically implies $(I, F) \times C$ for some $C \in \mathcal{H}$ then
11:         Let $C \in \mathcal{H}$ be the most recently added constraint
               for which it holds for that $\mathcal{G}$ logically implies $(I, F) \times C$
12:         $C' = (I, F) \times C$
13:     else
14:         $C' \leftarrow (I, F)$
15:     end if
16:     $\mathcal{H} \leftarrow \mathcal{H} \cup \{Crit_{\mathcal{G}}(C')\}$
17: end while
18: return $\mathcal{H}$
```

Figure 3.1: **Algorithm** GAVLearn

The promised GAVLearn algorithm is shown in Figure 3.1. The input of the GAVLearn algorithm include a goal GAV mapping $\mathcal{G}$ whose specification is unknown and a set E of universal examples for $\mathcal{G}$. The GAVLearn algorithm is an iterative learning algorithm. In each iteration, the GAVLearn algorithm checks if the current candidate mapping $\mathcal{H}$ fits the input set E of data examples. If yes, $\mathcal{H}$ is returned; otherwise, the algorithm will construct a counterexample $(I, \texttt{can-sol}_{\mathcal{H}}(I))$ obtained from the data example $(I, J) \in E$ such that $\mathcal{H}$ does not fit (I, J). In fact we will show that it must be the case that $J \subsetneq \texttt{can-sol}_{\mathcal{H}}(I)$. Subsequently, the GAVLearn algorithm will compute a candidate constraint from (I, F) where F is a fact in the set $J \setminus J'$. The candidate constraint (I, F) is informative for the learning, because it intuitively describes a transformation that is not captured by the current mapping $\mathcal{H}$ but is captured by the goal mapping. It follows that a new GAV constraint will be extracted from (I, F) and be added to $\mathcal{H}$.

We next show that the GAVLearn algorithm is an Occam algorithm that leads to a PAC learning algorithm for GAV($\mathbf{S}$,$\mathbf{T}$). We have the following theorem.

Theorem 3.9 Assume a goal schema mapping $\mathcal{G}$ of size at most n. GAVLearn is an Occam algorithm. In fact, given an oracle access to $\mathcal{G}$ and given a set E of canonical universal examples for $\mathcal{G}$, GAVLearn returns a GAV schema mapping $\mathcal{H}$ that fits E and the size of $\mathcal{H}$ is at most n^2.

Proof. Let us denote by $\mathcal{H}_i$ the value of the variable $\mathcal{H}$ after the i-th iteration of the **while** loop of GAVLearn, where $\mathcal{H}_0 = \emptyset$, and $\mathcal{H}_i$ is undefined if i is larger than the total number of iterations of the **while** loop; let us also denote by C_i the GAV constraint that was added at the i-th iteration, that is, $\mathcal{H}_i = \mathcal{H}_{i-1} \cup \{C_i\}$. For every GAV schema mapping $\mathcal{H}$, and for every $C \in \mathcal{G}$, let $\mathcal{H}(C) = \{C' \in \mathcal{H} \mid C \to C'\}$.

We first prove the following claim that is an adaptation of a lemma in [21].

Claim 3.10 For every $i \geq 0$ such that $\mathcal{H}_i$ is defined, the following statements are true.

(a) $\mathcal{H}_i$ consists of GAV constraints that are sound with respect to $\mathcal{G}$. In particular, the canonical solution $\texttt{can-sol}_{\mathcal{H}_i}(I)$ must be a solution for I w.r.t. $\mathcal{H}_i$ but not a solution for I w.r.t. $\mathcal{G}$. Moreover, we have that $\texttt{can-sol}_{\mathcal{H}_i}(I) \subsetneq J$ (where J is the target instance of the data example (I, J) obtained at Line 6 of GAVLearn).

(b) If $i > 0$, then there is no $C \in \mathcal{H}_{i-1}$ such that $C \to C_i$, that is, $\mathcal{H}_{i-1}$ does not logically imply C_i.

(c) If $C \in \mathcal{G}$, then the set $\mathcal{H}_i(C)$ is either empty or has a minimal element with respect to the homomorphism preorder $\to$ (i.e., there is an instance $I \in \mathcal{H}_i(C)$ such that for every instance $I' \in \mathcal{H}_i(C)$ we have that $I \to I'$).

The Claim 3.10(a) is trivially true for $\mathcal{H}_0 = \emptyset$ because, in this case, $\texttt{can-sol}_{\mathcal{H}_0}(I)$ is an empty instance and thus $(I, \texttt{can-sol}_{\mathcal{H}_0}(I))$ trivially satisfies $\mathcal{H}_0$ and does not satisfy $\mathcal{G}$ (note that $\mathcal{G}$ cannot be empty if GAVLearn reaches Line 6). For $i > 0$, we know that, by definition of GAVLearn, $\texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$ is a solution for I w.r.t. $\mathcal{H}_{i-1}$. It is easy to see that $\texttt{can-sol}_{\mathcal{H}_{i-1}}(I) \neq J$ because otherwise (I, J) would not be chosen at Line 6 of GAVLearn. We remain to show that $\texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$ is not a solution for I w.r.t. $\mathcal{G}$ and $\texttt{can-sol}_{\mathcal{H}_{i-1}}(I) \subsetneq J$. We show it by contradiction. Suppose that $\texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$ is a solution for I w.r.t. $\mathcal{G}$, then we have that $J \subset \texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$ (because J is the canonical solution of I w.r.t. $\mathcal{G}$). Since $\texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$ is the canonical universal solution of I w.r.t. $\mathcal{H}_{i-1}$. The instance J cannot be a solution for I w.r.t. $\mathcal{H}_{i-1}$. However, this contradict the fact that $\mathcal{H}_{i-1}$ is sound w.r.t. $\mathcal{G}$. This also shows that $\texttt{can-sol}_{\mathcal{H}_{i-1}}(I) \subsetneq J$. Therefore, (a) is proved.

For Claim 3.10(b). Let $I, \texttt{can-sol}_{\mathcal{H}_{i-1}}(I), F$, and C' be as computed in the i-th iteration of GAVLearn. By construction, we know that $\texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$ is a solution for

I w.r.t. $\mathcal{H}_{i-1}$ but not a solution for I w.r.t. $\mathcal{G}$. Since $F \notin \texttt{can-sol}_{\mathcal{H}_{i-1}}(I)$, we have that $\mathcal{H}_{i-1}$ does logically implies (I, F), that is, there is no $C \in \mathcal{H}_{i-1}$ such that $C \to (I, F)$ (by Lemma 3.6). Moreover, by a well-known basic properties of direct products, there is no $C \in \mathcal{H}_{i-1}$ such that $C \to C'$; Consequently, since $C_i = Crit_{\mathcal{G}}(C')$, which is a subinstance of C', there is no $C \in \mathcal{H}_{i-1}$ with $C \to C_i$.

For Claim 3.10(c). By induction on i. Let $C \in \mathcal{G}$. If $C_i \notin \mathcal{H}_i(C)$, then the result follows from the induction hypothesis. Therefore, let $C_i \in \mathcal{H}_i(C)$, that is, $C \to C_i$. We will show that, in this case, C_i is a minimal element of $\mathcal{H}_i(C)$ with respect to the homomorphism preorder $\to$. We distinguish two cases:

(1) There is no $C_j \in \mathcal{H}_{i-1}$ such that $C \to C_j$. In this case, C_i is the only GAV constraint in $\mathcal{H}_i$ into which C maps and hense the result hods trivially.

(2) There are $C_j \in \mathcal{H}_{i-1}$ such that $C \to C_j$. By induction hypothesis, there is such a $C_j \in \mathcal{H}_{i-1}$ that is minimal with respect to the homomorphism preorder, that is, $C_j \to C_k$ for all $C_k \in \mathcal{H}_{i-1}$ that have the property that $C \to C_k$. In particular, by previous item, we have that C_j is the most recently added GAV constraint with the property that $C \to C_j$. Let I, J, F, and C' be as computed in the i-th iteration of the algorithm. From the description of the algorithm and from the previous remarks, we have that $C_i = Crit_{\mathcal{G}}((I, F) \times C_k)$ for some C_k that was added no less recently than C_j. Since $C \to C_i$, we have that $C \to C_k$. Hence, C_j was added no less recently than C_k. In fact, this means that $C_j = C_k$ and hence $C_i = Crit_{\mathcal{G}}((I, F) \times C_j)$. It follows that $C_i \to C_j$ and we can infer that C_i is a

minimal element of $\mathcal{H}_i(C)$ with respect to the homomorphism preorder $\to$. This concludes the proof for laim 3.10(c).

Claim 3.11 Let C' and $Crit_{\mathcal{G}}(C')$ be as computed in the i-th iteration of GAVLearn. The size of $Crit_{\mathcal{G}}(C')$ is at most n (recall that n is the size of $\mathcal{G}$).

The proof for Claim 3.11 is straightforward. Recall that $Crit_{\mathcal{G}}(C')$ is critically sound with respect to some constraint $T \in \mathcal{G}$, which means there is a homomorphism $h : T \to Crit_{\mathcal{G}}(C')$. In fact we can see that the homomorphism h is surjective, for otherwise, $Crit_{\mathcal{G}}(C')$ would not have been a critical sound sub-constraint of T. It follows that $Crit_{\mathcal{G}}(C')$ is a homomorphic image of T such that the number of atoms on the left-hand side of $Crit_{\mathcal{G}}(C')$ is bounded by the number of atoms on the left-hand side of T. Since the size of T is bounded by n, the size of $Crit_{\mathcal{G}}(C')$ is bounded by n.

For every $i \geq 1$, define s_i to be $\Sigma_{T \in \mathcal{G}} s_i^T$, where s_i^T is 0 if $\mathcal{H}_i(T)$ is empty, and the number of variables occurring in the minimal element of $\mathcal{H}_i(T)$, otherwise (this is well defined by Claim 3.10(c)). That the schema mapping $\mathcal{H}$ returned is fitting for E follows directly from the definition of the algorithm. We still need to show that the algorithm terminates after at most n many iterations (recall that n denotes the size of $\mathcal{G}$, that is, the number of variables occurring in the constraints of $\mathcal{G}$), and the size of $\mathcal{H}$ returned has size at most n^2. We first make the following claim.

Claim 3.12 For every $i \geq 1$, $s_{i+1} > s_i$.

We now prove Claim 3.12. Assume that $\mathcal{H}_{i+1} = \mathcal{H} \cup \{C_{i+1}\}$. Since C_{i+1} is critically sound with respect to $\mathcal{G}$, there is some $T \in \mathcal{G}$ such that $T \to C_{i+1}$. By Claim 3.10(2) and 3.10(3), C_{i+1} is the minimal element of $\mathcal{H}_{i+1}(T)$. It follows that s_{i+1}^T is the domain size of C_{i+1}. If $\mathcal{H}_i(T)$ is empty, then claim holds true trivially. Otherwise, let C_i be its minimum element. We know that $C_{i+1} \to C_i$. In fact, the homomorphism from C_{i+1} to C_i must be surjective, in other words, C_i must be the homomorphic image of T, for otherwise, we could obtain a nonsurjective homomorphism from T to C_i (via $T \to C_{i+1}$), contradicting, via Lemma 3.6, the fact that C_i is critically sound with respect to $\mathcal{G}$. We also know by Claim 3.10(2) that T is not isomorphic to C_i. It follows that the domain size of C_{i+1} is larger than that of C_i and Claim 3.12 is proved.

It follows that Claim 3.12 shows that GAVLearn terminates after at most n many iterations, because s_i is at most n. Note that if for some well defined s_i such that s_i is greater than n, it means that there is a constraint $c \in \mathcal{H}$ such that c is the minimum element of $\mathcal{H}(T)$ for some $T \in \mathcal{G}$ and the domain size of c is greater than that of T. Since there is a homomorphism from T to c, the homomorphism in question is not surjective, and that contradicts the fact that c is critically sound with respect to T. Therefore, GAVLearn terminates after at most n many iterations.

Claim 3.13 The size of the mapping $\mathcal{H}$ returned by GAVLearn is bounded by n^2.

The proof of Claim 3.13 is immediately followed from Claim 3.11 and the fact that GAVLearn terminates after at most n many iterations (which we just proved).

The above argument concludes the proof of Theorem 3.9. □

Although not explicitly stated, GAVLearn is in fact an active learning algorithm that, given a set E of universal examples, generates a number of new universal examples from E that guide the learning algorithm towards critically sound GAV constraints w.r.t. the goal mapping $\mathcal{G}$. Concretely, as shown in Lemma 3.7, when computing a critical sound sub-constraint $Crit_{\mathcal{G}}(C')$ from C', the algorithm would repeatedly query the labeling oracle (i.e., the goal mapping $\mathcal{G}$) with sub-instances obtained from I, and that would generate $O(|I|)$ (where $|I|$ denotes the number of facts in I) many new universal examples for $\mathcal{G}$. These additional set of universal examples plays an important role in the learning.

The following example illustrates the workflow of the GAVLearn algorithm.

Example 3.14 Let $\mathcal{M}_g = \{\mathbf{S}, \mathbf{T}, \Sigma\}$ be a GAV schema mapping that we are trying to learn (i.e., the goal schema mapping), where $\mathbf{S}= \{S, R, M, N\}$, $\mathbf{T}= \{T, Q\}$, and Σ is the following set of constraints.

$$\Sigma = \{S(x,y) \land R(y,z) \rightarrow T(x,z), M(x,y) \land N(y,z) \rightarrow Q(x,y,z)\}.$$

For illustration purpose, the specification of $\mathcal{M}_g$ is revealed. Let E be a set of universal examples for $\mathcal{M}_g$, and E consists of a single data example (I, J), where

$$I = \{S(a,b), R(b,c), M(a,b), N(b,c)\} \textit{ and } J = \{T(a,c), Q(a,b,c)\}.$$

There are many GAV schema mappings for which (I, J) is universal. Consider two GAV mappings $\mathcal{M}_1 = \{\mathbf{S}, \mathbf{T}, \Sigma_1\}$ and $\mathcal{M}_2 = \{\mathbf{S}, \mathbf{T}, \Sigma_2\}$, where

$$\Sigma_1 = \{M(x,y) \land N(y,z) \rightarrow T(x,z), \quad S(x,y) \land R(y,z) \rightarrow Q(x,y,z)\}, \textit{ and}$$

$$\Sigma_2 = \{M(x,y) \land N(y,z) \land S(x,y) \land R(y,z) \rightarrow T(x,z),$$
$$M(x,y) \land N(y,z) \land S(x,y) \land R(y,z) \rightarrow Q(x,y,z)\}.$$

The example (I, J) is universal for both $\mathcal{M}_1$ and $\mathcal{M}_2$, but these two mappings are quite different from the goal mapping $\mathcal{M}_g$. Both $\mathcal{M}_1$ and $\mathcal{M}_2$ would not generalize well on future unseen universal examples of $\mathcal{M}_g$. For instance, if we are provided with a new source instance $I' = \{S(u, v), R(v, z), M(x, y)\}$, then $\mathcal{M}_1$ would produce $\{Q(u, v, z)\}$ and $\mathcal{M}_2$ would produce $\emptyset$. It follows that the former would generate an unexpected fact (i.e., $Q(u, v, z)$) and the latter would miss an expected fact (i.e., $T(u, z)$). We next show that how GAVLearn learns a GAV mapping from the present schema-mapping discovery scenario.

Suppose that we are executing GAVLearn from scratch. Initially, we have an empty candidate mapping $\mathcal{H} = \emptyset$. In the first learning iteration, GAVLearn would check if $\mathcal{H}$ and $\mathcal{M}_g$ *agree* on E. Note that, given a data example (I, J), we say two schema mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ agree on (I, J) if $\texttt{can-sol}_{\mathcal{M}_1}(I) = \texttt{can-sol}_{\mathcal{M}_2}(I)$. We say two mappings agree on a set E of data example if the two mappings agree on every example of E. The algorithm would detect that $\mathcal{H}$ and $\mathcal{M}_g$ disagree on (I, J). It follows that GAVLearn would construct a data exmaple (I, J'), where J' is obtained by chasing I with respect to $\mathcal{H}$. In particular, J' would be an empty instance because $\mathcal{H}$ is an empty schema mapping. Note that J' is a solution for I w.r.t. $\mathcal{H}$ (but not a solution for I w.r.t. $\mathcal{M}_g$). Then GAVLearn would select a fact f that is in J but not in J'. Let us assume that $f = T(a, c)$ is selected. Subsequently, GAVLearn would extract a new GAV constraint from $(I, \{f\})$ using the computation described in Lemma 3.7. Concretely, the following procedure illustrates the computation using Lemma 3.7.

$\{\cancel{S(a,b)}, R(b,c), M(a,b), N(b,c)\} \xrightarrow{\mathcal{M}g} \{Q(a,b,c)\}$: keep $S(a,b)$

$\{S(a,b),\cancel{R(b,c)}, M(a,b), N(b,c)\} \xrightarrow{\mathcal{M}g} \{Q(a,b,c)\}$: keep $R(b,c)$

$\{S(a,b), R(b,c),\cancel{M(a,b)}, N(b,c)\} \xrightarrow{\mathcal{M}g} \{T(a,c)\}$: remove $M(a,b)$

$\{S(a,b), R(b,c),\cancel{M(a,b), N(b,c)}\} \xrightarrow{\mathcal{M}g} \{T(a,c)\}$: remove $N(a,b)$

When the computation terminates, the algorithm would convert the simplified instance $(I', \{f\})$, where $I' = \{S(a,b), R(b,c)\}$ and $f = T(a,c)$, into a GAV constraint $S(x,y) \wedge R(y,z) \rightarrow T(x,z)$ that would be added to $\mathcal{H}$.

In the second learning iteration, we have that $\mathcal{H} = \big(\mathbf{S}, \mathbf{T}, \{S(x,y) \wedge R(y,z) \rightarrow T(x,z)\}\big)$. The two mappings $\mathcal{H}$ and $\mathcal{M}_g$ still do not agree on (I, J). This time, a new instance (I, J'), where $J' = \{T(a,c)\}$, would be constructed and a new fact $Q(a,b,c)$ would be selected. By the similar procedure described above, a new constraint $M(x,y) \wedge N(y,z) \rightarrow Q(x,y,z)$ would be added to $\mathcal{H}$ at the end of the second iteration.

In the third iteration, $\mathcal{H}$ and $\mathcal{M}_g$ agree on T. Therefore, the algorithm would terminate and return $\mathcal{H}$, where

$$\mathcal{H} = \big(\mathbf{S}, \mathbf{T}, \{S(x,y) \wedge R(y,z) \rightarrow T(x,z), M(x,y) \wedge N(y,z) \rightarrow Q(x,y,z)\}\big).$$

We can see that $\mathcal{H}$ is identical to $\mathcal{M}_g$. $\square$

From Occam Learnability to PAC Learnability

Note that Theorem 3.9 shows that GAVLearn is an Occam algorithm with $\alpha = 0$ and $k = 2$ (recall Def. 3.1). Theorems 3.9 and 3.2 imply the following corollary.

Corollary 3.15 Assume a goal GAV schema mapping $\mathcal{G}$ of size at most n. Let ϵ and δ be two rational numbers in $(0, 1)$, and let E be a set of m random universal examples for $\mathcal{G}$ obtained according to some probability distribution D. Let $\mathcal{H}$ be the GAV mapping returned by GAVLearn taking E and $\mathcal{G}$ as input. When

$$m \geq \left(\frac{n^2 \ln 2 + \ln(2/\delta)}{\epsilon} \right),$$

then with probability at least $1 - \delta$, we have that $\texttt{ERR}_{\mathcal{G},D}(\mathcal{H}) \leq \epsilon$.

Here, $\texttt{ERR}_{\mathcal{G},D}(\mathcal{H})$ denotes the true error rate of the mapping $\mathcal{H}$ w.r.t. $\mathcal{G}$ and D. Concretely, $\texttt{ERR}_{\mathcal{G},D}(\mathcal{H}) = \mathbf{Pr}_{x \in D}(\mathcal{G}(x) \neq \mathcal{H}(x))$ where $\mathbf{Pr}_{x \in D}(\mathcal{G}(x) \neq \mathcal{H}(x))$ is the probability of the event $\mathcal{G}(x) \neq \mathcal{H}(x)$ when x is a source instance drawn from the distribution D, and $\mathcal{G}(x)$ (resp. $\mathcal{H}(x)$) represents $\texttt{can-sol}_{\mathcal{G}}(x)$ (resp. $\texttt{can-sol}_{\mathcal{H}}(x)$). Corollary 3.15 shows that GAVLearn is a PAC learning algorithm.

Note that GAVLearn does not run in polynomial time because computing the canonical universal solution for a source instance I w.r.t. a mapping $\mathcal{H}$ is not in PTIME, when both I and $\mathcal{H}$ are part of the input. However, if GAVLearn is provided with an oracle for NP (used for evaluating the candidate mapping $\mathcal{H}_i$ on the input set E of universal examples), then GAVLearn is an efficient PAC learning algorithm for GAV(**S**,**T**).

3.3 Experimental Evaluation

In this section, we present an experimental evaluation of GAVLearn using mapping scenarios created by iBench, which is a primitive-based integration metadata generator [9]. We also compared GAVLearn with two existing methods: (1) the GAV fitting algorithm, denoted by GAV-Fitting, which is a special case of the GLAV fitting algorithm introduced in [3], and (2) The TALOS system introduced in [68].

Relevant parameters of iBench. iBench can be used to create schema mapping scenarios used for evaluating schema mapping systems. iBench defines a mapping scenario as a pair of a source and target schema and a mapping between the schemas. The mapping in question is specified by primitives, where each primitive is a simple integration scenario. iBench permits intricate control over the mapping scenario generation process, which allows the user to create mapping scenarios of different characteristics using various parametrized primitives. The following primitives in iBench are relevant to GAV constraints.

- `COPY` primitive (copying).
  $$\text{E.g., } S(x) \rightarrow T(x).$$
- `DelAttr` primitive (delete attributes).
  $$\text{E.g., } S(x,y) \rightarrow T(x).$$
- `Merge` primitive (join multiple source atoms to create one target atom).
  $$\text{E.g., } S_1(x,y) \wedge S_2(y,z) \rightarrow T(x,y,z).$$

Moreover, iBench also provides the following relevant configuration options to further refine the primitives being constructed:

- `JoinSize` (number of joining source atoms per merge primitive);
- `SourceShare` and `TargetShare` (percentage of primitives that share source and target relations);
- `JoinKind` (type of joins, such as star and chain, used to construct merge primitives).

Note that there are other parameters (e.g., number of attributes per source relation) that can be tuned to create different GAV mapping scenarios. We left them with the default values since they are less important than the ones listed above. The settings of the above primitives and configurations define the "shape" of a GAV mapping scenario. Once we fix a mapping scenario $\mathcal{S}$, iBench can then produce a random mapping specification (i.e., a set of GAV constraints) conforming to $\mathcal{S}$. Moreover, once a mapping specification is finalized, users can ask iBench to create a random source instance for the mapping specification in question. One relevant parameter is `NumOfElements` that specifies the number of facts (per source relation) to be created for the random source instance in generation.

Mapping scenarios. In our evaluation, we created three GAV mapping scenarios: *simple, moderate*, and *complex*. The three scenarios were created using different parameter settings and configurations (more details will be provided shortly). For each mapping scenario $\mathcal{M}$, we create a number of test cases where each consists of a random mapping specification S that conforms to $\mathcal{M}$ and a number of universal examples for $\mathcal{M}$.

Mapping specification generation. In our evaluation, the size (i.e., number of GAV constraints) of a random mapping specification is determined by the mapping scenario it conforms to. Concretely, a random mapping specification conforming to the simple (resp. moderate and complex) scenario consists of 10 (resp. 20 and 30) GAV constraints. In each mapping specification, 30% (resp. 30% and 40%) of its GAV constraints are `COPY` (resp. `DelAttr` and `Merge`) constraints. For instance, if S is a random mapping specification conforming to the moderate scenario, then 6 (resp. 6 and

8) constraints of S are `COPY` (resp. `DelAttr` and `Merge`) constraints. For all scenarios, in order to create a set of non-trivial mapping specifications, we also set both `SourceShare` and `TargetShare` to 100%[3]. We left `JoinKind` with its default value, which is star join. Apart from the fact that the three scenarios include a different number of constraints, we set different `JoinSize` values for them. Concretely, the simple (resp. moderate and complex) scenario has `JoinSize` $= 3$ (resp. 6 and 9). Clearly, more constraints and a higher `JoinSize` lead to a more complicated mapping scenario.

Data example generation. We generate universal examples by leveraging iBench. Creating a random universal example for a specific mapping specification S by iBench consists of two steps: (1) request a random source instance I conforming to the source schema of S by specifying the parameter `NumOfElements`, and (2) compute the $\texttt{can-sol}_S(I)$ using S. We remark that, in the first step, iBench would generate a source instance in which every source relation has exactly n facts if `NumOfElements` $= n$. This is counter-intuitive since such kind of data examples are not common in practice. To address this, we introduce a sampling parameter α used for creating random sub-instances of the source instances produced by iBench. Given a source instance $I = \{f_1, \ldots, f_m\}$ created by iBench and given a sampling ratio α%, we then create a new instance I' from I such that every $f_k \in I$ has α% chance to be selected into I'. It follows that the deliberate introduction of randomness into the generation process can result in more natural data examples. As we will show later, the value of the sampling parameter α turned out to have significant influence on the "quality" of the resulting source instance. Intuitively, if α is low,

[3]We observed that the actual percentage of shared source and target relations in generated mapping specifications are around 40%-50%

Table 3.1: Statistics of mapping scenarios

	Copy	**DelAttr**	**Merge**	**JoinSize**
simple	3	3	4	3
moderate	6	6	8	6
complex	9	9	12	9

In each scenario, we consider different combinations of α and ExNum, where α ={0.05,0.1,0.3} and ExNum={10,30,50,70,90}

the resulting source instance may not have "sufficient" information for the learning task.

We can repeat the above procedure m times to create m random universal examples, and the number of universal examples is thus another tunable parameter of our evaluation. As observed in the work of Alexe et al [3], data examples of size comparable to the size of constraints of the goal schema mapping are common in a user-interactive schema mapping design environment. Recall that the maximum number of constraints of our mapping scenarios is 30. In our evaluation, the number of universal examples, i.e., `ExNum` (which are later split into 70% training and 30% evaluation), generated for each random mapping specification ranges from 10 to 90. An overview of the statistics of the three mapping scenarios is provided in Table 3.1.

Implementation. GAVLearn was implemented in Java. The underlying relational database engine is PostgreSQL v9.3.10. All experiments were run on a 64-bit Linux machine with a Intel i7-4770 CPU (3.40GHz) and 20GB RAM. PostgreSQL engine was run with default settings.

3.3.1 Evaluation Methodology

Prior works evaluate the quality of a GAV schema mapping according to the concept of "correctness" (e.g., [3, 21, 38]), that is, a GAV schema mapping $\mathcal{M}$ is said to be correct if, for a given set E of data examples, the set E is universal for $\mathcal{M}$. In our evaluation, we use *F-score* that measures the quality of a GAV schema mapping with respect to a set E of examples by a real number in $[0,1]$. The F-score of a schema mapping $\mathcal{M}$ with respect to a set E of data examples is defined according to the precision and recall of $\mathcal{M}$ with respect to E. We first define some concepts and notations needed.

True positives, false positives, and false negatives. Assume a GAV schema mapping $\mathcal{M}$ and a data example (I,J) where $J=\{f_1,\ldots,f_n\}$. Let $J'=\{f'_1,\ldots,f'_m\}$ be $\texttt{can-sol}_{\mathcal{M}}(I)$. We say that a fact $f'_k \in J'$ is a *true positive* fact of $\mathcal{M}$ w.r.t. (I,J) if $f'_k \in J$; a fact $f'_k \in J'$ is a *false positive* fact of $\mathcal{M}$ w.r.t. (I,J) if $f'_k \notin J$; a fact $f_k \in J$ is a *false negative* fact of $\mathcal{M}$ w.r.t. (I,J) if $f_k \notin J'$ (see Figure 3.2 for an illustration).

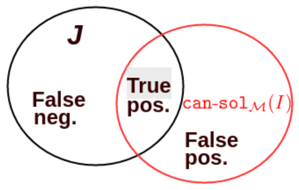


Figure 3.2: True positives, false negatives, and false positives

Precision and recall. Assume a GAV schema mapping $\mathcal{M}$ and a data example (I,J). We define the *precision* and *recall* of $\mathcal{M}$ for (I,J) as

$$Prec_{(I,J)}(M) = \frac{TP(\mathcal{M})}{TP(\mathcal{M})+FP(\mathcal{M})}, \quad Recall_{(I,J)}(M) = \frac{TP(\mathcal{M})}{TP(\mathcal{M})+FN(\mathcal{M})}$$

where $TP(\mathcal{M})$ (resp. $FP(\mathcal{M})$, and $FP(\mathcal{M})$) is the number of true positive (resp. false positive, and false negative) facts of $\mathcal{M}$ w.r.t. (I,J). We then define the F-score of a schema mapping $\mathcal{M}$ as follows.

$$F\text{-}score_{(I,J)}(\mathcal{M}) = \frac{2\times Prec_{(I,J)}(\mathcal{M})\times Recall_{(I,J)}(\mathcal{M})}{Prec_{(I,J)}(\mathcal{M})+Recall_{(I,J)}(\mathcal{M})}.$$

We can see that F-score is a combined measure that takes into account both precision and recall. In order to have a high F-score, a learned mapping must achieve both high precision and high recall. Note that the above definitions can be naturally extended to the case of multiple data examples.

Methodology. We evaluate GAVLearn both in terms of quality (i.e., F-scores) and in terms of runtime efficiency. We are also interested in comparing GAVLearn with other methods in terms of the two aspects. When evaluating the quality of a schema mapping, we use the Hold-Out method that is widely used in the machine learning field. Concretely, in the Hold-Out method, the labeled data examples are randomly divided to three subsets: a *training set*, which is used to learn a model; a *validation set*, which is used to fine tune model's parameters and select the best-performing model; a *test set*, which is a hold-out set of examples used to estimate the future performance of a model. Note that not all machine learning evaluations need a validation set. In our evaluation, we generate a number of universal examples for each test case, and then we randomly split these examples into a training set (70%) and a hold-out test set (30%). Note that the ratio of (70%:30%) is commonly used in most machine learning evaluations. Our evaluation methodology is formally summarized as follows:

Fix a mapping scenario $\mathcal{S}$, for every sampling ratio $\alpha \in \{0.05, 0.1, 0.3\}$ (the choices of sampling ratios will be discussed later) and for every integer $\texttt{ExNum} \in \{10, 30, 50, 70, 90\}$, we do the following.

a) Create a random GAV mapping specification $\mathcal{M}$ conforming to $\mathcal{S}$ using iBench.

b) Create `ExNum` many random universal examples for $\mathcal{M}$ with sampling ratio α and `NumElements` $= 20$.

d) Randomly split the examples obtained in step b) into a training set E (70%) and a hold-out test set V (30%).

e) Learn a GAV mapping $\mathcal{M}_\ell$ from E, and compute the precision and recall of $\mathcal{M}_\ell$ on V.

To lower the inaccuracy caused by a possible outlier, we performed three random runs for each test case. We report the average precision and recall of the three random runs. Moreover, we report an overall F-score computed from the average precision and recall. We run 45 different tests since there are three scenarios, where each considers three different sampling ratios and five different numbers of universal examples. Note that each test case can be seen as a triple (E, V, $\mathcal{M}$), where E (resp. V) is a training set (resp. a hold-out test set) and $\mathcal{M}$ is the goal GAV schema mapping that we are trying to learn.

Measure the Goodness of A Training Set

We would like to evaluate the quality of the mappings returned by GAVLearn. Note that, in general, the "quality" of a training set has a significant influence on the outcome of the learning tasks. In a precise sense, the quality of a training set determines the best solution that an algorithm can produce. Therefore, it would be good to have a measure that indicates the "best" mapping can be produced by any GAV mapping learning algorithm in a particular test case. If we have an idea of what is the best result one can produce, then we can see how "close" the mappings returned by GAVLearn (and other works) are to the best result.

We introduce the measure *Degree of Comprehensiveness (DoC)* that can be used to measure the quality a training set E in a test case (E, V, $\mathcal{M}$). Let $\mathcal{M} = \{\mathbf{S}, \mathbf{T}, \Sigma = \{c_1, \ldots, c_n\}\}$ be a GAV schema mapping specification, where Σ is a set of GAV constraints. Let E (resp. V) be a training set (resp. the corresponding hold-out test set) created for $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ as described above. We define the degree of comprehensiveness of E with respect to V and $\mathcal{M}$ as

$$\mathrm{DoC}_{\mathcal{M},V}(E) = \frac{|\Sigma^{\mathrm{E}} \cap \Sigma^{\mathrm{V}}|}{|\Sigma^{\mathrm{E}}|},$$

where Σ^x denotes the subset of Σ that is triggered in the set x (we say a constraint c is triggered in E if there is an example $(I, J) \in E$ such that there is a homomorphism from the left-hand side of c to I). The DoC value can indicate the goodness of a training set with respect to a held-out test set. It follows that the DoC vallue somewhat determines the best mapping an algorithm can produce. Observe that if $\mathrm{DoC}_{\mathcal{M},V}(E) = 100\%$, then E provides sufficient information for the learning task since for every constraint t of $\mathcal{M}$ such that t is triggered in V, we have that t is also triggered in E. In this case, theoretically, there could be some learning algorithm that exactly identifies Σ^{E} (not Σ). On the other hand, if $\mathrm{DoC}_{\mathcal{M},V}(E) = 50\%$, then it intuitively indicates that the hold-out test set V can trigger substantially more (i.e., $|\Sigma^{\mathrm{E}}|$ many more) constraints of $\mathcal{M}$ than E does. It follows that, in this case, we cannot expect that there is a learning algorithm that exactly identifies Σ^{E}. The above discussion also implies that the DoC value of a training set E has a significant influence on the recall of GAV mapping learned from E.

Going back to the choices of sampling ratios, when we created the universal examples, we noticed that that if the sampling ratio was greater than 0.4, then a training set consisting of only 7 universal examples would have a DoC value of 1.0 for all mapping scenarios. Therefore, we used three relatively small sampling ratios such that we could generate training sets of different DoC values, and use them to test GAVLearn.

3.3.2 Evaluation of GAVLearn

Highlights of the results of executing GAVLearn on the three scenarios are provided in Tables 3.2, 3.3 and 3.4. The results of test cases of the simple scenario, where the sampling ratio (i.e., α) equals to 0.3, are omitted, because the GAVLearn algorithm achieved an F-score of 1 in all these test cases. Each row of the three tables corresponds to a particular test case. In the rest of the section, we use the expression "$\texttt{X}$-$\alpha(\theta)$-$n(\beta)$ " to refer to the test case in scenario $\texttt{X}$ with $\alpha = \theta$ and $n = \beta$, where α is the sampling ratio and n is the total number of universal examples (including both training and hold-out examples). Since we used a 70%-30% split, the size of the training set (i.e., $|E|$) of a test case of size n equals to $n \times 0.7$. For example, $\texttt{Complex}$-$\alpha(0.0.5)$-$n(10)$ refers to the first row of Table 3.4, that is, the test case of the complex scenario, where the sampling ratio is 0.05 and the number of universal examples is 10. For each test case, we report the following measures of the mappings returned by GAVLearn (recall that every test case has three random runs): the overall F-score ($\texttt{F}_\text{s}$), the average recall ($\overline{\text{Rec}}$), the average precision ($\overline{\text{Prec}}$), the average DoC ($\overline{\text{DoC}}$), the average number of iterations ($\overline{\text{Iter}}$) performed by GAVLearn, and the average running time ($\overline{\text{Time}}$). We also report

Table 3.2: Results of running GAVLearn on simple scenario

α	n	$\|E\|$	$\mathrm{F_s}$	$\overline{\text{Rec}}$	$\overline{\text{Prec}}$	$\overline{\text{DoC}}$	$\overline{\text{Iter}}$	$\overline{\text{Time}}$
	10	7	0.68	0.51±35%		0.55	7.7	3.9s
	30	21	0.88	0.78±20%		0.77	10.3	10.6s
0.05	50	35	0.94	0.90±13%	1	0.88	11.7	20.6s
	70	49	0.91	0.83±22%		0.83	11	27.1s
	90	63	1	1		1	13	36s
	10	7	0.85	0.74±20%		0.72	9	7.6s
	30	21	0.91	0.84±8%		0.71	11	22.3s
0.1	50	35	1	1	1	1	13	38.5s
	70	49	1	1		1	13	54.7s
	90	63	1	1		1	13	1m6.7s

Table 3.3: Results of running GAVLearn on moderate scenario

α	n	$\|E\|$	$\mathrm{F_s}$	$\overline{\text{Rec}}$	$\overline{\text{Prec}}$	$\overline{\text{DoC}}$	$\overline{\text{Iter}}$	$\overline{\text{Time}}$
	10	7	0.54	0.37±27%	1	0.389	12.3	3.9s
	30	21	0.80	0.69±10%	0.95±7%	0.80	19	10.6s
0.05	50	35	0.94	0.89±6%	1	0.89	19.6	20.6s
	70	49	0.96	0.96±1%	0.96±4%	0.94	25	27.1s
	90	63	1	1	1	1	25	36s
	10	7	0.77	0.63±10%		0.61	15.7	7.6s
	30	21	0.97	0.94±4%		0.94	23.7	22.3s
0.1	50	35	1	1	1	1	25	38.5s
	70	49	1	1		1	25	54.7s
	90	63	1	1		1	25	1m6.7s
	10	7	1	1		1	25	26s
	30	21	0.98	0.98±2‰		0.93	26.33	1m6s
0.3	50	35	0.99	0.99±1‰	1	0.93	29	1m51s
	70	49	0.99	0.99±2‰		0.93	32.33	2m55s
	90	63	0.99	0.99±0‰		0.95	31	3m51s

the standard deviation values of precision and recall computed from three random runs of each test case.

Analysis of Results

By varying the sampling ratio and number of universal examples, we were able to create a number of training sets with different DoC values. Recall that the DoC value of a training set E can indicate the best mapping that any GAV mapping learning algorithm can produce.

As shown in Tables 3.2, 3.3 and 3.4, the GAVLearn algorithm produced

Table 3.4: Results of running GAVLearn on complex scenario

α	n	$\lvert E\rvert$	$\mathrm{F_s}$	$\overline{\mathrm{Rec}}$	$\overline{\mathrm{Prec}}$	$\overline{\mathrm{DoC}}$	$\overline{\mathrm{Iter}}$	$\overline{\mathrm{Time}}$
	10	7	0.50	0.35±6%		0.33	13.3	8.7s
	30	21	0.85	0.75±6%		0.67	17	24.4s
0.05	50	35			1		25	44s
	70	49	1	1		1	25	58.5s
	90	63					25	1m19s
	10	7	0.78	0.64±22%		0.64	16.3	17.6s
	30	21					25	51s
0.1	50	35	1	1	1	1	25	1m31s
	70	49					25	2m6.5s
	90	63					25	2m50s
	10	7	0.97	1	0.96±3%	1	43	1m7s
	30	21	0.98	0.98±2%	0.98±2%	1	46.33	2m41s
0.3	50	35	1	1	1	1	47.67	4m55s
	70	49	0.98	0.99	1	1	55.67	8m6.6s
	90	63	0.99	0.99	1	1	53.67	10m9s

mappings that achieved very high precision values (at least 95%) in all test cases. We also observed that, among those cases where the precision values were not 100%, the variances of precision values were low (at most 7%). These observations conform to our expectation. Since, by definition, the GAVLearn algorithm iteratively produces GAV constraints that are critically sound with respect to the goal mapping, we expect the produced GAV mappings to be high-precision. In contrast, the average recall values varied significantly from 0.37 to 1. We also observed an evident dependency of the recall values on the corresponding DoC values. As we can see in Tables 3.2, 3.3 and 3.4, the recall value of a test case is highly similar to the corresponding DoC value. Recall that the DoC value of a training set (w.r.t. the corresponding hold-out test set) has a significant influence on the recall of the mapping produced by a learning algorithm. In some sense, the similarity between recall values and the corresponding average DoC values indicates that, the mappings returned by GAVLearn achieved recall values that were close to the best recall values that could have been achieved by a GAV learning algorithm. Moreover, an interesting observation is that, when the number of training examples was low (e.g., seven), the variance of recall was high (e.g., 35% in the Simple-$\alpha(0.05)$-$n(10)$). The variance of recall reduced significantly when more training examples were provided to GAVLearn. Moreover, by fixing a sampling ratio, the DoC value grows as the number of universal examples increases. In most cases, when the DoC value of a training set was 1, then the recall value was also 1. There were only three cases, in the complex scenario, where the DoC values were 1 but the recall values were not 1 (such as the test case Complex-$\alpha(0.3)$-$n(30)$). Although, in these three cases, the recall values were not 1, the recall values were still high (at least 0.98) together with low variances ($\leq 2\%$). Therefore, the recall values of GAVLearn were high when the DoC values were high. Since the precision values were high ($\geq 95\%$) with low variances and the recall values were highly similar to the DoC values, the F-scores of GAVLearn mainly depended on the recall values. Therefore, the similarity between F-scores and DoC values, as shown in Tables 3.2, 3.3 and 3.4, indicates that GAVLearn is stable and effective.

We next discuss the performance of GAVLearn in terms of several other aspects:

Size of training set. We noticed that better results can be achieved by providing GAVLearn with more data examples (for a fixed sampling ratio). More training examples provided the learning algorithm with more source facts that may increase the DoC value of the training set, and/or increase the number of homomorphisms from the left-hand sides of the constraints of the goal mapping to the training set. In either case, it would provide the learning algorithm with more information, which would in turn lead to a better mapping.

Active learning. As already pointed out in Section 3.2, GAVLearn is an active learning algorithm that systematically generates a large number of guided universal examples from the provided training set. These additional universal examples are extremely important for the learning because they will guide the algorithm to critically sound constraints with respect to the goal mapping. By Lemma 3.7, we know that the number of new examples generated during the computation of critically sound constraints is linear in the size of E (i.e., the total number of facts contained in E). For example, in the test case `Complex`-$\alpha(0.3)$-$n(90)$, every example of the training set contains about 320 source facts, meaning that the training set contains about $20K \approx 63 * 320$ source facts. Therefore, the mapping returned by GAVLearn, in the `Complex`-$\alpha(0.3)$-$n(90)$ test case, was actually an outcome of learning from $20K$ universal examples. These additional examples contributed a lot to the high-quality result (i.e., an F-score of 0.99).

Runtime efficiency. The runtime of the GAVLearn algorithm is reported in Tables 3.2, 3.3 and 3.4 (see the last column). The runtime of GAVLearn depended mainly on the size of the training set. Since GAVLearn is an active learning algorithm that generates a large number of universal examples, it further increases the runtime of GAVLearn. It will be a bottleneck if the training set is very large. However, our experiments were designed to simulate a common user-interactive schema mapping design environment, where the number of data examples needed is usually comparable to the size of constraints of the goal schema mapping. Therefore, the runtime of GAVLearn is reasonable.

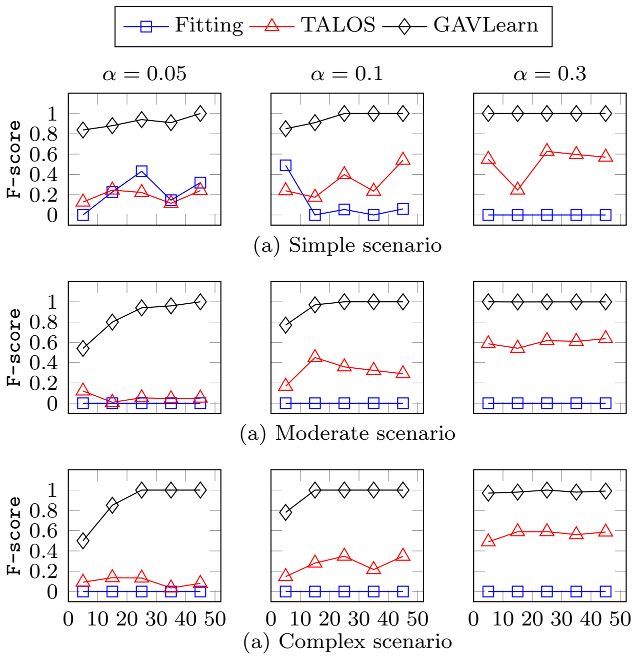


Figure 3.3: Comparison of GAV-Fitting, TALOS, and GAVLearn

3.3.3 Comparison with Existing Methods

In this section, we compare GAVLearn with two existing methods: the GAV-Fitting algorithm [3] and the TALOS system using the aforementioned three mapping scenarios. We did not implement the GAV-Fitting algorithm because executing GAV-Fitting on those test cases is trivial (the reason will be clear shortly), and the authors of [68] shared with us the implementation of the TALOS system. Figure 3.3 reports the quality (in terms of F-score) of GAV mappings returned by the three algorithms with respect to the three scenarios (X-axes are number of training examples). We can see that GAVLearn produced GAV mappings with much higher F-scores in all test cases than the other two. TALOS was the sub-optimal, and GAV-Fitting produced the worst results where all F-scores were close to zeros. We next discuss the differences among them in detail.

GAV-Fitting. In [3], the author proposed the GAV-Fitting algorithm that, given a set E of ground data examples, generates the canonical GAV fitting schema mapping for E if there is a fitting GAV mapping for E. In particular, GAV-Fitting is a special case of the GLAV fitting algorithm proposed in [3]. GAV-Fitting is a two step algorithm that first checks if there exists a fitting GAV schema mapping for E using a procedure called homomorphism extension test (see [3] for more details). If the answer is yes, then the canonical GAV fitting schema mapping of E will be returned; otherwise, it reports "none exists" and terminates. In our setting, since the training set E is a set of universal examples for the goal mapping. This means that there is always a GAV fitting schema mapping for E, so the GAV-Fitting algorithm will always return the canonical GAV schema mapping of E. By construction, the canonical GAV schema mapping of E is obtained by spelling out the facts of E. Hence, it will not work well on the hold-out test set V. More specifically, the canonical GAV schema mapping obtained from E would achieve high precision but low recall on V, because the canonical GAV mapping obtained from E would produce a non-empty target instance only if one of the source instances of V has a homomorphic image of one of the source instances of E. In machine learning, the mappings returned by GAV-Fitting are considered to be overfitting. That is why the F-scores of the GAV-Fitting algorithm were so low.

TALOS. The TALOS system achieved much higher F-scores in almost every test case of the three mapping scenarios than GAV-Fitting did. The core technique of TALOS is a classification algorithm based on a decision tree. Concretely, the algorithm re-organizes the constants of the input database by a decision tree such that multiple instance-equivalent Union-Select-Projection-Join (SPJU) queries (i.e., GAV mappings) can be extracted from the decision tree with respect to the query result provided. Once the decision tree is ready, the TALOS system will then return one, among all candidate queries (i.e., mappings), whose cost is minimum (see [68] for more details). One of the advantages of TALOS over GAV-Fitting is that TALOS takes into account the "succinctness" of a learned model. Therefore, it will return a succinct model that is less specific to the input database (i.e., the training set in our setting).

As shown in Figure 3.3, with more training examples and a higher sampling ratio, the F-score of the GAV mappings produced by TALOS became higher. We noticed obvious improvements (in terms of F-scores) when we increased the sampling ratio from 5% to 10% (as well as from 10% to 30%). However, these F-scores were still lower than the corresponding F-scores of GAVLearn. We manually inspected the queries (i.e., GAV mappings) returned by TALOS. We found that a substantial portion of these queries included constants belonging to the training set E (these constants were used in the selection conditions of the resulting SPJ queries). This means that TALOS still has an overfitting issue.

Runtime efficiency. The GAV-Fitting algorithm is the fastest algorithm among the three because its execution is trivial. The TALOS system is also very efficient. In particular, all experiments of TALOS finished within 2 seconds. Note that TALOS learns from the provided training set only, whereas GAVLearn learns from both the provided training set and the large number of additional universal examples generated on the fly. GAV-Fitting and TALOS are much faster than GAVLearn, but the quality of their results is worse.

Summary. The main difference between GAVLearn and the other two algorithms is that, GAVLearn uses the provided labeling oracle to generate a large number of guided universal examples. Our experiments showed that the additional guided universal examples were extremely important for GAVLearn because, if the guided universal examples are not available, GAVLearn is essentially the GAV-Fitting algorithm. We note that both GAV-Fitting and TALOS focus on producing a GAV mapping that perfectly fits the training set. In other words, learning a GAV mapping that will generalize well on future universal examples is not considered by their objective functions. In contrast, the GAVLearn algorithm focuses on producing a GAV mapping that is semantically close to the goal mapping with respect to both the training set and future unseen data. This explains the superiority of GAVLearn over the other two methods in our evaluation.

3.4 Conclusions

In this chapter, we presented the GAVLearn algorithm that is a PAC learning algorithm for GAV schema mappings. The work was built on the learning framework introduced in [21]. We implemented the GAVLearn algorithm and evaluated it using mapping scenarios created by iBench. We show experimentally that the GAVLearn algorithm can produce high-quality GAV schema mappings. As a byproduct of the experimental evaluation, we also introduced a new measure (i.e., the F-score measure) to evaluate the quality of schema mappings with respect to a set of data examples. One important question that remains to be answered is the learnability of LAV schema mappings (likewise for GLAV schema mappings).

Part II

Learning Rule-Based Entity Resolution

Chapter 4

Learning Rule-Based Entity Resolution Algorithms from Data Examples

In this chapter, we present an example-driven active learning system, ER-LEARN, that learns high-quality rule-based entity resolution algorithms at scale. Note that the definition of data examples in the setting of entity resolution is different from that in the setting of schema-mapping discovery. In the setting of schema-mapping discovery, a data example is a pair (I, J), where I is a source instance and J a target instance. In the setting of entity resolution, a data example is a pair (r, s), where r and s are two records of the input datasets. Moreover, in the setting of ER, data examples will be labeled as matches or non-matches by a human user. The goal of our work is to learn rule-based entity resolution algorithms that are both high-precision and high-recall from data examples. An ER rule `R` is considered to be high-precision if most of the links, i.e., pair of records, returned by `R` are indeed true matches; an ER rule `R` is considered to be high-recall if R covers a significant fraction of the true matches in the datasets. It follows that a high-quality rule-based ER algorithm should consist of multiple high-precision ER rules such that the combined recall of these rules is high.

As discussed in Chapter 1, ERLEARN actively searches for two types of data examples: (1) likely false positive examples that are used to improve the precision of current rule and (2) likely false negative examples that are not covered by the current rule but are likely matches. The latter type of examples are more challenging to identify but have a tremendous impact in refining an existing rule towards better recall and in finding additional good rules. One of our main contributions is the ability to generate likely false negatives that in turn, enable learning at scale of multiple ER rules, each having significant coverage (i.e., recall) of the space of true matches.

Roles of Different Types of Examples

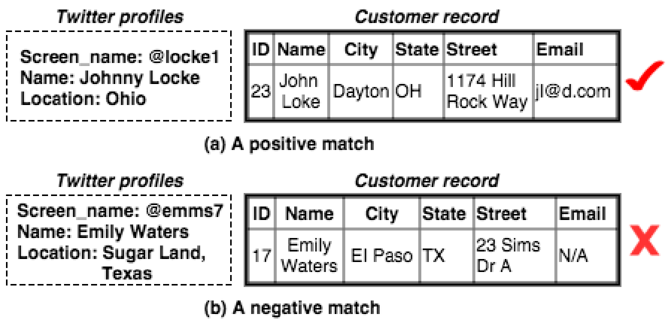
In order to show the roles that different types of examples (i.e., false positives and false negatives) play in our learning system, consider the following matching scenario, which follows from our Twitter-Customer scenario discussed in Chapter 1. Assume that the system has learned an initial rule, `Rule1` shown below.

```
match Twitter T, Customer C by Rule1:
   T.lastname = C.lastname
   AND firstNameMatch(T.firstName, C.firstName)
   AND T.location.state = C.state
```

Figure 4.1: An ER rule

Assume that a new false positive example, shown in Figure 4.2(b), has just been found.

This example is a false positive, with respect to `Rule1`, because it satisfies all the



Twitter profiles

Screen_name: @locke1
Name: Johnny Locke
Location: Ohio

Customer record

ID	Name	City	State	Street	Email
23	John Loke	Dayton	OH	1174 Hill Rock Way	jl@d.com

✓

(a) A positive match

Twitter profiles

Screen_name: @emms7
Name: Emily Waters
Location: Sugar Land, Texas

Customer record

ID	Name	City	State	Street	Email
17	Emily Waters	El Paso	TX	23 Sims Dr A	N/A

✗

(b) A negative match

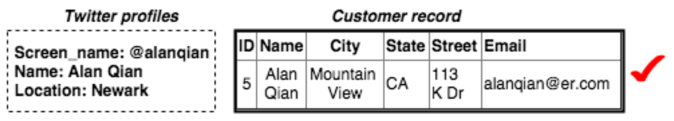
Figure 4.2: Examples of matches: (a) positive, (b) negative.

conditions of `Rule1`, but the user labels it as a negative match. As a result of seeing such example, `Rule1` can be automatically refined into a more conservative (and more precise) rule such as `Rule1′` below. This rule adds an extra condition specifying that it only applies to people whose last names are not in the top 80% frequent last names in the United States. The refined rule has an increased precision (and, in particular, avoids false positive examples such as the one just seen). False positive examples are relatively easier to find, since they live within the result set of the current rules.

```
match Twitter T, Customer C by Rule1′:
   T.lastname = C.lastname
   AND firstNameMatch(T.firstName, C.firstName)
   AND T.location.state = C.state
   AND lastNameFrequencyFilter(C.lastName, 80)
```

Figure 4.3: Example: ER rule `Rule1′` refined from `Rule1`

In contrast, likely false negative examples are examples of matches that are not covered by any of the existing rules. Figure 4.4 shows a false negative example for our scenario; this represents a match that is not covered by the existing rule (`Rule1′`) but the user would label it as positive.



Twitter profiles

Screen_name: @alanqian
Name: Alan Qian
Location: Newark

Customer record

ID	Name	City	State	Street	Email
5	Alan Qian	Mountain View	CA	113 K Dr	alanqian@er.com

✓

Figure 4.4: A false negative example for `Rule1′`.

Once the learning algorithm becomes aware of such examples, a second rule such as `Rule2` shown below may be generated (in addition to `Rule1′`). The false negative examples enable the learning algorithm to form new combinations of matching conditions and exploit new attributes (e.g., email and Twitter screen handle, in this case).

```
match Twitter T, Customer C by Rule2:
  T.lastname = C.lastname
  AND firstNameMatch(T.firstName, C.firstName)
  AND sameHandle(T.screen_name, C.emailHandle)
```

Figure 4.5: Another ER rule

Finding false negatives is challenging, simply because the search space is much larger (e.g., for our Twitter-Customer scenario, a single high-precision rule may yield a few thousands or tens of thousands of matches, while the search space of likely false negatives is as large as the remaining portion of the cross-product of the two datasets, i.e., 10^{16} pairs). To address this challenge, we have developed a novel search heuristic, which we call *Rule-Minus*, that actively searches for likely false negative examples, by systematically removing one or more conditions from an existing rule. As we show in the rest of the chapter, this heuristic is quite effective in increasing the space of examples that is "seen" by the learning algorithm.

In the rest of the chapter, we describe ERLEARN in detail. In Section 4.1, we introduce the language of entity resolution rules. In Section 4.2, we formalize the entity resolution learning problem. In Section 4.3, we describe our learning system in details including the system architecture and learning methodology. In Section 4.4, we present a comprehensive experimental evaluation of our system. In Section 4.6, we give a summary of our work and discuss future work.

4.1 The Language of ER Rules

In this section, we give the formal definition of the language of ER rules. We first introduce several notations and concepts.

Without loss of generality, we assume that, unless otherwise stated, the ER task in consideration has two input datasets, denoted by D_1 and D_2. We will use (r, s) to denote a pair of records (in short, *a pair*) in the cross product of D_1, D_2, that is, $(r, s) \in D_1 \times D_2$, where $r \in D_1, s \in D_2$.

Matching predicates are the basic building blocks of ER algorithms. In our approach we consider various types of matching predicates, where these predicates range from basic equalities on attributes of records, to more complex (similarity or threshold-based) comparison functions, and can also make use of filters and normalization functions. In fact, learning the “right” compositions of the aforementioned functions is one of the central tasks of our learning approach. Before giving the definition of matching predicates, we first define the following key ingredients used in matching predicates.

A *matching function* f is a boolean function that can apply directly on a pair

(r, s) or on components of r and s (e.g., projections, or other values extracted from r and s).

Example 4.1 The equality test `r = s` is a matching function that applies directly on a pair (r, s); $\texttt{sim(}s_1\texttt{,}s_2\texttt{,}\theta\texttt{)}$ is another matching function which tests if the similarity score between two strings is equal to or greater than a given value θ. □

We also use *unary functions*, which are (non-boolean) functions that can apply directly on an individual record r (resp. s) or on a component of r (resp. s). Unary functions can be composed to obtain other unary functions.

Example 4.2 The projection `T.firstname` is a unary function that extracts the "first-name" value from a Twitter profile T; the function `UpperCase(x)` is a normalization function that converts every character of the string `x` to its uppercase. Another unary function is the composition of the previous two functions (e.g., `UpperCase(T.firstname)`). □

We can now define matching predicates. Given a set of matching functions $F = \{f_1, \ldots, f_n\}$ and a set of unary functions $U = \{u_1, \ldots, u_m\}$, a *matching predicate* is defined as:

$$P(r, s) = f_k\left(\underbrace{u_0(r) \circ \cdots \circ u_i(r)}_{\texttt{arg1}},\ \overbrace{u'_0(s) \circ \cdots \circ u'_j(s)}^{\texttt{arg2}},\ \theta\right),$$

where: (1) $(r, s) \in D_1 \times D_2$, (2) `arg1` (resp. `arg2`) is a composition of unary functions applied on r (resp. s), (3) θ is an optional fixed number (e.g., a threshold) needed for the case when the matching function f_k expects such argument. In a matching predicate, `arg1` and `arg2` are optional, but cannot both be missing. Specifically, if one of the two arguments is missing, then we sometimes call the resulting predicate a filter predicate (since it applies only on one of the input datasets). The main difference between matching functions and matching predicates is that matching predicates are actual instantiations of matching functions on concrete "paths" within the input records. Matching predicates also contain concrete instantiations of the numerical parameter θ.

Example 4.3 In the predicate `UpperCase(T.lastname) = UpperCase(C.lastname)`, the matching function is the equality test `=`, and the two arguments are obtained via the composite unary functions `UpperCase(T.lastname)` and `UpperCase(C.lastname)`. As an example of a different nature, assume that `lastNameFrequencyFilter (x,`θ`)` is a function that returns true if its first argument `x` is a last name that is not in the top θ% of the population in terms of last name frequency. In general, there could be infinitely many possible values for the numerical parameter θ. Instead of learning the exact value of θ over a full continuous range (e.g., between 0 and 100% in this case), we use an approximation strategy to make the space of possible values of θ finite: we discretize the space of possible parameter values by using certain granularities. For example, if the granularity is 10, we instantiate the matching function `lastNameFrequencyFilter (x,`θ`)` into multiple matching predicates: { `lastNameFrequenceFilter (x, 10), ...,` `lastNameFrequencyFilter (x, 90)` }, where the value of the parameter θ is built into the matching predicate. The learning algorithm will then have the role to identify the right matching predicate (with the right parameter value). □

A *blocking predicate* is defined as a conjunction of one or more matching predicates that has the additional property that it "logically" partitions the input datasets into blocks of records so that the subsequent ER algorithms compare only records within the same block. Blocking is an essential feature in entity resolution that is needed to avoid comparing pairs over the cartesian product $D_1 \times D_2$. While learning of blocking predicates may be a task in itself, in this paper, we assume that blocking predicates are marked explicitly by a human user. As an example, the predicate `UpperCase(T.lastname) = UpperCase(C.lastname)` may be explicitly marked as a blocking predicate. If this predicate is used in an ER rule, the effect is that we will compare only records whose last names are identical (modulo uppercase conversion).

We now define an entity resolution (ER) rule R, with respect to D_1, D_2, via a conjunction of matching predicates:

```
match  $D_1$ $r$,  $D_2$ $s$  by
    R:  $P_1(r,s)$ AND ... AND $P_m(r,s)$,
```

such that at least one blocking predicate appears in the conjunction. An entity resolution (ER) algorithm A is then defined via a disjunction of rules:

```
match  $D_1$ $r$,  $D_2$ $s$  by
    R$_1$:   $P^1_1(r,s)$ AND ... AND $P^1_{m_1}(r,s)$,
    ...
    R$_k$:   $P^k_1(r,s)$ AND ... AND $P^k_{m_k}(r,s)$.
```

The *result* of applying an ER rule R to datasets D_1 and D_2 is the set of all pairs (r, s) that satisfy the conjunction of matching predicates of R. We sometimes use the term *links* of R, denoted by *links*(R), for the result of R on D_1 and D_2. The result of applying an ER algorithm A is the union of pairs (r, s) that are produced by the individual rules of `A`. Thus, a pair (r, s) is in the result of `A` if it satisfies the disjunction of the conjunctions of matching predicates in `A`.

A Connection to the Language of GAV Constraints. Recall that schema mappings are expressed in the language of GLAV constraints. The language of GLAV constraints and the language of ER rules are two different languages for two different information integration problems. However, under appropriate assumptions, we can establish a connection between the two languages. Without loss of generality, we may assume that each record of the input datasets of an ER task has a unique ID. The result of an ER algorithm A can be encoded as a binary relation consisting of pairs ($r1$, $r2$), where r_1 and $r2$ are IDs of two records, which are identified as a match by A. Let us denote the resulting binary relation by Link($id1$, $id2$). It follows that the goal of entity resolution is thus to populate the target Link($id1$, $id2$) relation from the input datasets (note that, ideally, the Link relation is expected to contain only the true matches among the input datasets). Therefore, the spirit of using schema mappings in order to translate a source instance to a target instance is similar to the spirit of using ER algorithms in order to create links among input datasets. It follows that, at a high level, every ER rule can be formulated as a GAV constraint in an extended language of GAV constraints. Take the ER rule `Rule1` shown in Figure 4.1 as an example. Assume that the Twitter data and the Customer data conform to the following schema:

Twitter(`id, firstname, lastname, nation, state`)

Customer(`id, firstname, lastname, state`).

Consider the following two constraints (shown in Figure 4.6).

$$\begin{aligned} t_1 : \forall \ldots\ & \text{Twitter}(\texttt{tid}, \texttt{tFN}, \texttt{LN}, \texttt{nation}, \texttt{state}) \wedge \text{Customer}(\texttt{cid}, \texttt{cFN}, \texttt{LN}, \texttt{state}) \\ & \wedge \underline{\texttt{firstNameMatch}(\texttt{tFN}, \texttt{cFN})} \rightarrow \text{Link}(\texttt{tid}, \texttt{cid}) \end{aligned}$$

$$\begin{aligned} t_2 : \forall \ldots\ & \text{Twitter}(\texttt{tid}, \texttt{tFN}, \texttt{cLN}, \texttt{nation}, \texttt{tState}) \wedge \text{Customer}(\texttt{cid}, \texttt{cFN}, \texttt{cLN}, \texttt{cState}) \\ & \wedge \underline{\texttt{equal}(\texttt{tLN}, \texttt{cLN})} \\ & \wedge \underline{\texttt{equal}(\texttt{tState}, \texttt{cState})} \\ & \wedge \underline{\texttt{firstNameMatch}(\texttt{tFN}, \texttt{cFN})} \rightarrow \text{Link}(\texttt{tid}, \texttt{cid}) \end{aligned}$$

Figure 4.6: `Rule1` as a GLAV-like constraint

We can see that t_1 and t_2 are both translated from `Rule1` shown in Figure 4.1, where the elements with underlines correspond to the matching functions of `Rule1`. The right-hand sides of t_1 and t_2 contain a single relational atom Link(`tic, cid`). We can see that the two constraints specify the same logic for obtaining links from the source data (i.e., the Twitter data and the Customer data). The only difference between t_1 and t_2 is how they translate the equality functions of `Rule1`. The constraint t_1 uses universal quantified variables to implicitly express the equality functions (over the two first name attributes and the two state attributes), whereas the constraint t_2 spells out the equality functions as well as the user-defined function (i.e., `firstNameMatch(tFN,cFN)`) as special elements. The two constraints above are syntactically similar to the language of GAV constraints. Obviously, t_1 and t_2 are not standard GAV constraints because of the existences of the special elements such as `firstNameMatch(tFN,cFN)`. However, by extending the standard language of GAV constraints with special expressions, such as arithmetic expressions and user-defined functions, we can use GAV-like constraints to define ER rules.

Although the above discussion shows that there is a connection between the two languages, we do not pursue designing a unified declarative language for expressing both schema mapping constraints and entity resolution rules. The discussion regarding the similarities of the two languages is meant to provide an intuition on how the two directions are related to each other.

4.2 Problem Definitions

In this section, we formulate the entity resolution learning problem. We first need several definitions and notations.

Specific to learning purposes, we will use the term *data examples* for pairs that are presented to a human user for labeling. We say a data example (r, s) is *positive* if it is identified as a *match* by a human user; otherwise, it is a *negative* example (we assume that a human user has perfect knowledge, so a real world match would be labeled as positive by a user, and negative otherwise).

Another set of terms highly used in this paper are *true positives, false positives*, and *false negatives* of a rule. A pair (r, s) is a true positive of R if $(r, s) \in links(\mathtt{R})$ and, furthermore, represents a true match (in particular, a human user would identify it as a match if shown[1]). A pair (r, s) is a false positive of R if $(r, s) \in links(\mathtt{R})$ but it is not a true match. A pair (r, s) is a false negative of R if $(r, s) \notin links(\mathtt{R})$ but it is a true match. The same definitions but with respect to an ER algorithm A are similarly defined by replacing R with A.

During the learning process, data examples are represented as feature vectors. Concretely, given a vector $\langle P_1, \ldots, P_m \rangle$ of distinct matching predicates that are applicable

[1]Note that the examples actually shown to a user will be a much smaller subset, in general.

on D_1, D_2, we define a *feature-augmented data example* $\widehat{F}(r,s)$, where $(r,s) \in D_1 \times D_2$, as follows

$$\widehat{F}(r,s) : \langle\ (r,s),\ P_1(r,s), P_2(r,s), \ldots, P_m(r,s)\ \rangle,$$

where the value of the k-th feature of $\widehat{F}(r,s)$ is the evaluation of P_k on (r,s) (for a concrete example, see Example 4.4). In our system, we use a simple algorithm that, given a collection F of matching functions and a collection U of unary functions, generates the matching predicates P that are applicable on D_1 and D_2. We do this by systematically enumerating matching predicates over F and U that are syntactically valid (i.e., satisfy the types of attributes in the input schemas, do not repeatedly apply the same unary function, etc.).

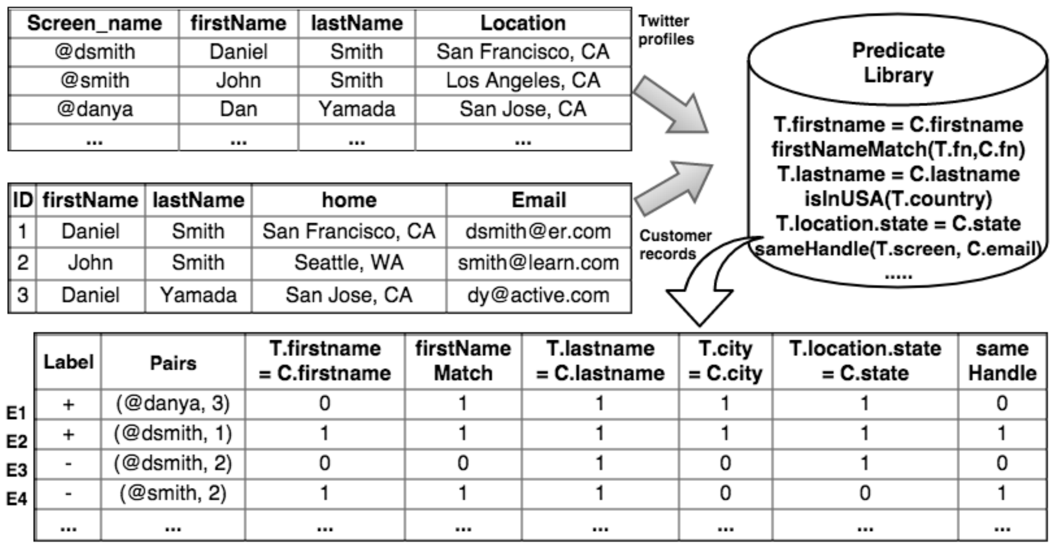
Example 4.4 Figure 4.7 illustrates the process of constructing and augmenting data examples from a catalog of applicable matching predicates (generated as discussed above). In our running scenario, data examples are obtained by pairing up Twitter profiles and customer records[2]; each pair is then augmented with features by evaluating all the matching predicates on the given pair. For each data example, the user will then add the label (shown in a separate column).

□

Precision and *recall* are two measures used in this paper to evaluate the ER algorithms returned by the active learning system. They are defined as

$$Prec(\mathtt{R}) = \frac{TP(\mathtt{R})}{TP(\mathtt{R})\ +\ FP(\mathtt{R})}\ , \quad Recall(\mathtt{R}) = \frac{TP(\mathtt{R})}{TP(\mathtt{R})\ +\ FN(\mathtt{R})}\ ,$$

[2]The process of obtaining such pairs is described in Section 4.3.



Screen_name	firstName	lastName	Location
@dsmith	Daniel	Smith	San Francisco, CA
@smith	John	Smith	Los Angeles, CA
@danya	Dan	Yamada	San Jose, CA
...	...	...	...

ID	firstName	lastName	home	Email
1	Daniel	Smith	San Francisco, CA	dsmith@er.com
2	John	Smith	Seattle, WA	smith@learn.com
3	Daniel	Yamada	San Jose, CA	dy@active.com

T.firstname = C.firstname
firstNameMatch(T.fn,C.fn)
T.lastname = C.lastname
isInUSA(T.country)
T.location.state = C.state
sameHandle(T.screen, C.email)
.....

	Label	Pairs	T.firstname = C.firstname	firstName Match	T.lastname = C.lastname	T.city = C.city	T.location.state = C.state	same Handle
E1	+	(@danya, 3)	0	1	1	1	1	0
E2	+	(@dsmith, 1)	1	1	1	1	1	1
E3	-	(@dsmith, 2)	0	0	1	0	1	0
E4	-	(@smith, 2)	1	1	1	0	0	1
	...	...	...	...	...	...	...	...

Figure 4.7: Feature-augmented data examples

where $TP(\texttt{R})$ is the number of true positives of R, $FP(\texttt{R})$ is the number of false positives of R, and $FN(\texttt{R})$ is the number of false negatives of R. Similar definitions apply for the case of an ER algorithm A.

We are now ready to define the entity resolution learning problem. Let τ be a precision threshold ($\tau \in [0, 1]$). The goal of the ER learning task is to learn an entity resolution algorithm A that

$$\textbf{\textit{maximizes}}\ Recall(\texttt{A})$$

$$\textbf{\textit{subject to:}}\ \forall \texttt{R} \in \texttt{A},\ Prec(\texttt{R}) \geq \tau.$$

In other words, the goal is to maximize the recall of A, while each entity resolution rule R in A has precision higher than τ.

4.2.1 Estimating Precision and Recall

In practice, especially on big data scenarios, computing the exact precision and recall of an ER rule is impractical. Computing the exact precision of an ER rule R requires one to examine every link of R. Given the fact that a rule may produce thousands of links, letting a human user label all these links is not feasible. Computing the exact recall of an ER rule R is even more difficult, because the search space of false negatives is much larger (and outside $links(\mathtt{R})$).

Estimating Precision. One widely accepted method of measuring precision is to estimate the precision of R on a randomly selected set of pairs from $links(\mathtt{R})$. In our approach, however, we will estimate the precision of R based on a carefully selected subset of links of R that have low similarity scores according to our active learning system. Intuitively, this represents an "adversarial" set of examples; if R has high precision with respect to this adversarial set, then it would have at least the same precision with respect to its entire coverage (or with respect to a randomly selected set of links). Thus, we use a very conservative method to measure the precision of R. See Section 4.3 for more details on how we obtain the "adversarial" set, and also Section 4.4 where we show experimentally that our estimation method gives very good results in practice.

A $\frac{1-\tau}{\tau}$-loss Estimation of Recall. Rather surprisingly, it turns out that for the above ER learning problem, where we need to maximize the recall subject to the precision constraints, we can estimate the recall of an ER rule R by simply counting the number of links of R. The following analysis shows, formally, why this is the case. By the earlier definition, we have that:

$$Recall(\mathtt{R}) = \frac{TP(\mathtt{R})}{TP(\mathtt{R})+FN(\mathtt{R})} = \frac{TP(\mathtt{R})}{matches(D_1,D_2)},$$

where $matches(D_1, D_2)$ is the number of all possible matches over $D_1 \times D_2$. Since the denominator is a constant, to compare the recall of two rules we only need to compare the numerators ($TP(\mathtt{R})$). If we now assume that each rule $\mathtt{R}$ satisfies the precision constraint, we have that

$$\tau \leq Prec(\mathtt{R}) = \frac{TP(\mathtt{R})}{TP(\mathtt{R})+FP(\mathtt{R})} = \frac{TP(\mathtt{R})}{LinkCount(\mathtt{R})},$$

where $LinkCount(\mathtt{R})$ is the size of $links(\mathtt{R})$. This implies that:

$$\frac{TP(\mathtt{R})}{\tau} \geq LinkCount(\mathtt{R}) \geq TP(\mathtt{R}) \geq \tau \cdot LinkCount(\mathtt{R}),$$

where the middle inequality follows from the fact that the true positives of $\mathtt{R}$ are a subset of $links(\mathtt{R})$. Thus, we can bound from above and below either $TP(\mathtt{R})$ or $LinkCount(\mathtt{R})$ using functions of the other, and thus $LinkCount(\mathtt{R})$ is a surrogate of $TP(\mathtt{R})$. In particular, we have that

$$TP(\mathtt{R}) \in [\tau \cdot LinkCount(\mathtt{R}), LinkCount(\mathtt{R})].$$

We can now bound the relative loss in recall, given that we estimate *Recall* via *LinkCount* in our optimization problem, as follows. Assume that we have two rules $\mathtt{R}_1$ and $\mathtt{R}_2$, both satisfying the precision constraint. Then, we have:

$$\frac{TP(\mathtt{R}_1)}{\tau} \geq LinkCount(\mathtt{R}_1) \geq TP(\mathtt{R}_1) \geq \tau \cdot LinkCount(\mathtt{R}_1)$$

$$\frac{TP(\mathtt{R}_2)}{\tau} \geq LinkCount(\mathtt{R}_2) \geq TP(\mathtt{R}_2) \geq \tau \cdot LinkCount(\mathtt{R}_2)$$

Suppose that the ER learning task estimates that $\mathtt{R}_1$ has better recall than $\mathtt{R}_2$ because $\mathit{LinkCount}(\mathtt{R}_1) > \mathit{LinkCount}(\mathtt{R}_2)$. Moreover, assume that, in fact, $\mathtt{R}_2$ has higher recall (i.e., $TP(\mathtt{R}_2) > TP(\mathtt{R}_1)$). The loss in recall (given that $\mathtt{R}_1$ was chosen over $\mathtt{R}_2$) is:

$$TP(\mathtt{R}_2) - TP(\mathtt{R}_1) \leq \mathit{LinkCount}(\mathtt{R}_2) - \tau \cdot \mathit{LinkCount}(\mathtt{R}_1) \leq$$

$$\mathit{LinkCount}(\mathtt{R}_1) - \tau \cdot \mathit{LinkCount}(\mathtt{R}_1) = \mathit{LinkCount}(\mathtt{R}_1)(1-\tau).$$

The relative loss on recall is then bounded as follows:

$$\frac{TP(\mathtt{R}_2) - TP(\mathtt{R}_1)}{TP(\mathtt{R}_1)} \leq \frac{\mathit{LinkCount}(\mathtt{R}_1) \cdot (1-\tau)}{\tau \cdot \mathit{LinkCount}(\mathtt{R}_1)} = \frac{1-\tau}{\tau},$$

which is a small number for large τ. Specifically, $\tau = 0.95 \Rightarrow \mathit{loss} \approx 5\%$; $\tau = 0.90 \Rightarrow \mathit{loss} \approx 11\%$. Note that one can control the error by changing the precision threshold τ. Furthermore, the above upper bound on relative loss is obtained by assuming worst cases. In fact, the ER rules we learn will often have precision that is higher than τ; thus, the actual error rate for recall is even lower.

4.3 Detailed System Architecture and Learning Methodology

In this section, we describe the design of our active entity resolution learning system in detail beginning with a high-level description of the general workflow. In particular, we describe how we iteratively discover and refine a set of ER rules where each iteration consists of: (1) automatic learning of a new candidate ER rule $\mathtt{R}$ together with active generation of new examples, followed by (2) the user interaction phase to

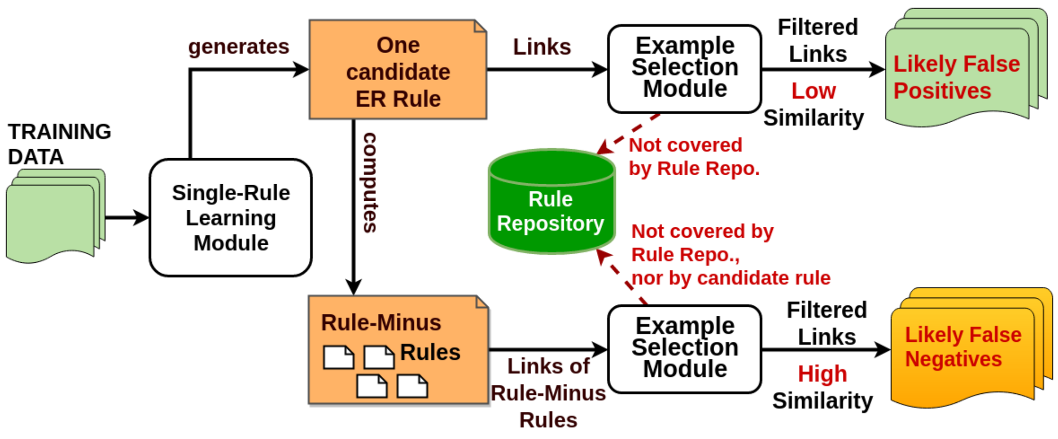


Figure 4.8: Automatic Rule Learning and Example Generation

label said examples and whether to accept or reject R.

Automatic Rule Learning and Example Generation

In the first phase, the system learns a single candidate rule R from the current training data set, and actively searches for both likely false positives of R and likely false negatives (of R and all rules that were previously accepted). The overall flow of this phase is illustrated in Figure 4.8.

The workflow can either be invoked with a small training set of pre-labeled examples (e.g., obtained by some process of exploration of the space of matches by a human user) or with an initial ER rule (provided by a domain expert). At the beginning of the first iteration, we also create an empty Rule Repository to be used as a database for storing the high-precision ER rules that will be accepted by the user. The first step in the workflow is the Single-Rule Learning Module that generates a single candidate ER rule based on the current training data. As an example, this module may generate

`Rule1` in Figure 4.1. Note that, in case the system was invoked by an initial rule then we simply use this as the candidate rule in the first iteration. Once the new candidate rule is available, the workflow splits into two branches: one for computing likely false positives with respect to the candidate rule (upper branch in Figure 4.8), and one for computing likely false negatives with respect to both the candidate rule and all the high-precision rules in the Rule Repository (lower branch in Figure 4.8).

To compute likely false positives (the upper branch in Figure 4.8), the learning system applies the candidate rule `R` on the actual input datasets and sends all the links of `R` to the Example Selection Module. To ensure the selection of only those examples that have high likelihood to be negative (since we are looking for likely *false* positives in this branch), the first step in the Example Selection Module (upper branch) is to discard all those links that are already covered by some rule in the Rule Repository. The underlying assumption here is that because each previously accepted ER rule in the Rule Repository is known to be above the precision threshold, any example covered by it is likely to be a match. The second step in the Example Selection Module is to select a few (e.g., 10) of the remaining links that have the *lowest similarity scores*. We use a similarity measure (defined in Section 4.3.2) to gauge the likelihood that a link is a match. The lower the similarity score is, the less likely it is that the link is a match, and vice versa. We refer to the links obtained in this branch as "likely false positive" examples of `R` because they satisfy the rule but may be negative given their low similarity scores. These examples represent good candidates of false positives (to be shown to the user). As an illustration, the pair in Figure 1.2(b) may be identified as a likely false positive.

On the other hand, in the branch of computing likely false negatives (the lower branch in Figure 4.8), the learning system first generates a set of "Rule-Minus" rules from the candidate rule `R`. Each Rule-Minus rule is obtained by removing matching predicates from `R`. Intuitively, each Rule-Minus rule is a mild generalization of `R` that enables the learning system to extend the space of potential matches beyond what is covered by `R`. Furthermore, each Rule-Minus rule is applied on the input datasets, and the resulting sets of links (one for each Rule-Minus rule) are sent to the Example Selection Module. In the Example Selection Module, the first step is to discard all the links that are covered either by `R` or by some already existing rule in the Rule Repository. Intuitively, we want to explore "new" examples that are not covered by `R` or by any of the rules in the repository. Among the surviving links, the Example Selection Module then selects a few examples that have *highest similarity scores.* We refer to the links obtained in this branch as "likely false negative" examples (with respect to both `R` and the rules in the repository). As an illustration, a pair such as the one in Figure 4.4 may be identified as a likely false negative.

Once the likely false positives and likely false negatives are obtained, the workflow moves to the user interaction phase.

User Interaction

In this phase, a human user is required to label the newly generated examples. Based on the user labeling, the system also decides whether to accept or reject the current candidate rule `R`. The flow of this phase is described in Figure 4.9.

We note that the sets of likely false positives and likely false negatives play

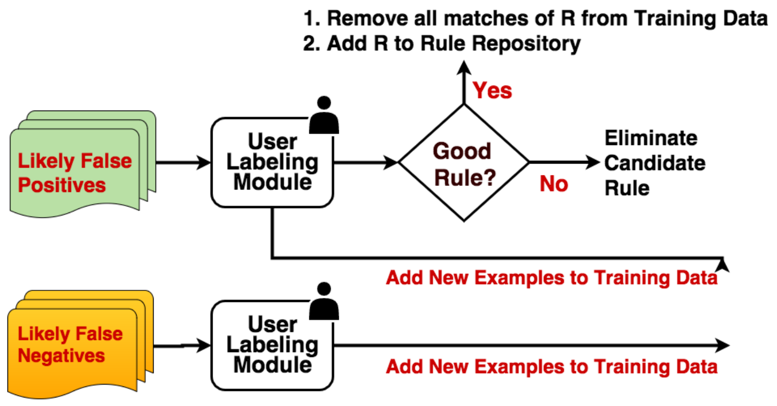

1. Remove all matches of R from Training Data
2. Add R to Rule Repository
Yes
Likely False Positives
User Labeling Module
Good Rule?
No
Eliminate Candidate Rule
Add New Examples to Training Data
Likely False Negatives
User Labeling Module
Add New Examples to Training Data

Figure 4.9: User Interaction Phase

different roles during the user interaction. On the lower branch of the flow, the likely false negatives are labeled by the user and then added to the training data as new augmented examples. Examples that are labeled as positive by the user in this branch are particularly important, since they may form new candidate rules in the next iterations. (This will become more apparent in Section 4.3.1.) For example, if the earlier likely false negative (the pair in Figure 4.4) is confirmed by the user as positive, a new ER rule may be generated (e.g., `Rule2` in Figure 4.5).

On the upper branch of the flow, the likely false positives are also labeled by the user and then added to the training data as new augmented examples. In addition, these likely false positives, which we recall are links of `R` with lowest similarity scores, are used to estimate the precision of `R`. In particular, the system determines whether or not the current candidate rule is high-precision according to the labels of these likely false positives. We distinguish two cases:

1. the candidate rule `R` is not good enough, which means its precision is lower than

the threshold τ on the set of likely false positives. In this case, we discard `R`.

2. the candidate rule `R` has precision higher than the threshold τ on the set of likely false positives. In this case, we add `R` to the Rule Repository, and then remove all matches of `R` from the training data. The last step is to ensure that in the next learning iteration, the Single-Rule Learning Module is able to produce a candidate rule other than `R`. (This is further explained in Section 4.3.4).

As mentioned in Section 4.2.1, the above method uses a conservative estimate of precision, since it checks the precision of `R` on an "adversarial" set consisting of links with low similarity scores. As a result, if `R` passes the threshold τ, it is likely to have even higher precision on a random sample of its links.

As an illustration, assume that the system rejects the candidate rule `Rule1` in Figure 4.1 because the likely false positives of `Rule1` contain many negative examples (e.g., the one shown in Figure 4.2(b)). In the next iteration, based on the availability of the new examples, the Single-Rule Learning Module will produce a different candidate rule, likely better (possibly a rule such as the earlier `Rule1`′). If this new candidate rule is of high-precision, then the system will accept it (and add it to the Rule Repository), remove all its matches from the training data, and go back to the Single-Rule Learning Module to learn other rules beyond the already accepted ones.

This learning process is repeated until either the user manually stops the system or the single-rule learning module does not produce any candidate rule that satisfies the precision constraint (to be discussed next).

4.3.1 Single-Rule Learning Module

The goal of the Single-Rule Learning Module is to learn a candidate ER rule from the available training data. Algorithm LearnRule is inspired by an exact learning algorithm for monotone DNF formulas introduced by Angluin [5]. In particular, if one interprets the vector $\langle P_1, \ldots, P_m \rangle$ of available matching predicates as a set of boolean variables, then our learning task amounts to finding a disjunction of rules, where each rule corresponds to a conjunction of (non-negated) boolean variables over $\langle P_1, \ldots, P_m \rangle$. While the classical Angluin algorithm requires an oracle for exact learning of the target formula, our LearnRule algorithm can be seen as a greedy adaptation that uses the training data instead of an oracle. Furthermore, different from the Angluin algorithm, our LearnRule algorithms learns a single "best" conjunction over $\langle P_1, \ldots, P_m \rangle$ that covers the most positive examples modulo the precision guarantee. The way in which we learn multiple conjunctions (rules) is via active learning, as part of the overall flow described earlier: once a rule is accepted, in the next iterations, we will rerun the same single-rule algorithm to learn the next "best" rule, given the new set of available examples. A description of our single-rule learning algorithm is given in Algorithm LearnRule.

Concretely, the LearnRule algorithm converts each positive example p in the training data T into an initial rule `R` by selecting all the matching predicates that evaluate to 1 on p. Initially, `R` is very specialized that possibly satisfies only one pair in the training data (p itself). The algorithm then simplifies `R` by systematically removing

Algorithm LearnRule(T, τ)

```
Input: $T$ - a set of labeled examples augmented by features;
    $\tau$ - precision threshold
 1: $POS \leftarrow$ all positive examples in $T$
 2: $NEG \leftarrow$ all negative examples in $T$
 3: $BestRule \leftarrow null$; $MaxContribution \leftarrow 0$
 4: $BestRule \leftarrow null$
 5: for every feature-augmented example $\widehat{F}(r,s) \in POS$ do
 6:     $candid \leftarrow \emptyset$
 7:     for every $val_i \in \widehat{F}(r,s)$ do
 8:         if $val_i = 1$ then //$val_i$ corresponds to evaluation $P_i(r,s)$
 9:             $candid \leftarrow$ add $\{P_i\}$ (the matching predicate)
10:         end if
11:     end for
12:     $(candid', contrib) \leftarrow$ Simplify($candid, T, \tau$)
13:     if $contrib > MaxContribution$ then
14:         $BestRule = candid'$; $MaxContribution = contrib$
15:     end if
16: end for
17: return $BestRule$
18:
19: procedure Simplify(candidRule, $T$, $\tau$)
20:     $ruleSet \leftarrow$ all possible subsets of $candidRule$
21:     $bestRule \leftarrow null$
22:     $maxContrib \leftarrow 0$
23:     for $rule \in ruleSet$ do
24:         if $Prec_T(rule) \geq \tau$ then
25:             $c \leftarrow$ # of matches in $T$ that are also links of $rule$
26:             if $c > maxContrib$ then
27:                 $bestRule = rule$; $maxContrib = c$
28:             end if
29:         end if
30:     end for
31:     return $(bestRule, maxContrib)$
32: end procedure
```

matching predicates to find a more general rule that produces more links on the training data (better coverage) possibly at the expense of decreased precision. After each simplification step, we ensure that the precision constraint is still satisfied by checking $Prec_T(\mathtt{R}) = |T^+ \cap links(\mathtt{R})|/|T \cap links(\mathtt{R})| \geq \tau$, where T^+ denotes the subset of positive examples of T, and τ is the input precision threshold. At the end, the system returns a single simplified rule $\mathtt{R}'$ covering the most positive examples in T among all rules satisfying the precision constraint on T.

Note that both positive and negative examples in the training data are used when calculating the precision of a simplified rule. Thus, negative examples play the role of eliminating low-precision simplified rules, and prevent the algorithm from simplifying a rule too much. For presentation purposes, the SIMPLIFY procedure described in LEARNRULE is a naive implementation where we enumerate subsets of the matching predicates. In our system, we implement an optimized version of SIMPLIFY, via a top-down exploration that stops early whenever we encounter low-precision rules. We also use dynamic programming techniques (e.g., to avoid recalculating the precision for a rule already explored).

Example 4.5 Consider a training data set consisting of the two positive examples (E_1, E_2) and the two negative examples (E_3, E_4) in Figure 4.7. Let the precision threshold τ be 90%. We illustrate Algorithm LEARNRULE in Figure 4.10.

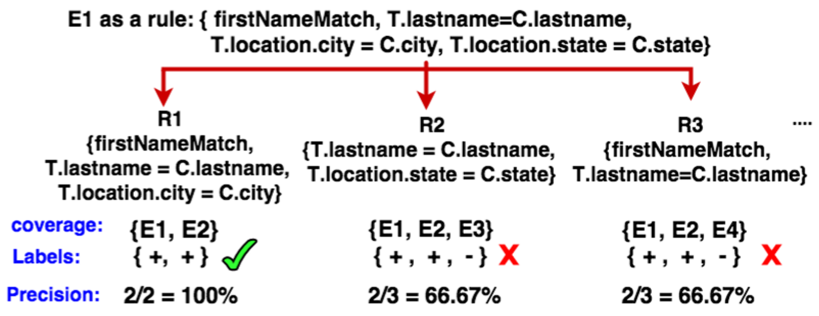

E1 as a rule: { firstNameMatch, T.lastname=C.lastname, T.location.city = C.city, T.location.state = C.state}
R1
{firstNameMatch, T.lastname = C.lastname, T.location.city = C.city}
R2
{T.lastname = C.lastname, T.location.state = C.state}
R3
{firstNameMatch, T.lastname=C.lastname}
....
coverage:
Labels:
Precision:
{E1, E2}
{ +, + }
2/2 = 100%
{E1, E2, E3}
{ + , + , - }
2/3 = 66.67%
{E1, E2, E4}
{ + , + , - }
2/3 = 66.67%

Figure 4.10: Generating and simplifying rules

The positive example E_1 is converted into an initial ER rule with four matching predicates. The algorithm then calls SIMPLIFY to generalize the initial rule by removing matching predicates. In the figure, we show three rules (i.e., `R1`, `R2`, and `R3`) that are simplifications of the initial rule. Computing the precision of `R1`, `R2`, and `R3` amounts to calculating the fraction of the covered examples that are labeled as positive by the user.

We see that `R1`, which is obtained from the initial rule by dropping the comparison on state, still satisfies the precision constraint ($\geq 90\%$). In contrast, `R2` and `R3` simplify too much and are discarded because of their low precision.

□

4.3.2 Finding Likely Misclassifications

We now give details on the computation of likely false positives and negatives for which we require an appropriately designed similarity measure. We first present the similarity measure and then describe how it is used to determine the likely misclassifications to be subsequently labeled by the user.

Similarity Measure

For the ensuing discussion, it is helpful to recall the notion of a weak learner or predictor [46]. A weak predictor is essentially a feature that performs barely better than random chance. Classic learning algorithms such as boosting aim to combine many weak predictors to produce a strong classifier [63]. Well known examples of weak predictors include 0-1 binary features and decision stumps [43]. For instance, the predicates `T.lastname = C.lastname` and `Jaccard_Similarity(T.name, C.name)` $\geq$ `90%` can be seen as such weak predictors (of links).

Our similarity measure does not need to be a perfect measure of how likely a pair is a duplicate. As we shall see subsequently, we only utilize the similarity measure to order the set of links returned by a rule. For our purposes, it suffices that some, not necessarily all, of the true matches appear at the top of the list obtained by sorting the links in decreasing order of the similarity measure and, conversely, some of the non matches appear at the bottom of the list. Note that we do not even need the ordering of the sorted links to be precise. Thus, we use a fixed combination of the matching predicates $\mathtt{P} = \langle P_1, \ldots, P_m \rangle$ (where each P_i may be interpreted as a decision stump or binary feature) as our similarity measure applicable on $(r, s) \in D_1 \times D_2$:

$$sim\,(r,s) = |\{\ P_k \mid P_k \text{ evaluates to 1 on } (r,s)\text{, where } P_k \in \mathtt{P}\}|$$

Intuitively, given a pair (r, s), $sim(r, s)$ counts the number of matching predicates that are true for (r, s) and assumes that the higher the count the more likely that it is a match. This is true if every matching predicate measures goodness of match (rather than its counter-positive). For instance, as opposed to `T.lastname = C.lastname`, `T.lastname` $\neq$ `C.lastname` does not measure goodness of match and we do not allow the latter.

Finding Likely False Positives

Given a candidate rule `R`, the goal of computing likely false positives of `R` is to find a small subset `LFP(R)` $\subseteq$ $links$(`R`) of size k, where k is a fixed constant, such that the links in `LFP(R)` are more likely to be labeled as negatives by a human user than those in $links$(`R`) $\setminus$ `LFP(R)`. The set `LFP(R)` is used to falsify a candidate rule `R` if `R` is not high-precision, and it also acts as the adversarial set, mentioned in Section 4.2.1, used to estimate the precision of `R`.

Given a candidate rule `R` and a constant k, we compute `LFP(R)` as follows. Let L be the set of links obtained from $links$(`R`) by discarding all links that are also links of a rule in the repository (as explained earlier, if a pair is covered by a previously accepted rule, it is very likely a match; however, the goal here is to look for negative examples). Then, `LFP(R)` is the subset of k links of L that have lowest similarity scores. The links in `LFP(R)` are likely false positives because they are links of `R` but they have higher likelihood to be negatives than $links(\texttt{R}) \backslash \texttt{LFP(R)}$, given their low similarity scores.

Note that if many of the links in `LFP(R)` are actually labeled as negatives by a user, once these newly labeled examples are added to the training data, the single-rule learning module will produce an ER rule other than `R` in the next iteration. This is because `R` will have lower precision with respect to the updated training data due to the existence of the newly added negative examples. Conversely, if at least $(\tau \cdot k)$ links of `LFP(R)`, where τ is the precision threshold, are actually positives, then we conclude that, with high likelihood, `R` achieves high precision over its entire coverage.

Finding Likely False Negatives

Let $\mathbf{R} = \{ \texttt{R}, \texttt{R}_1, \ldots, \texttt{R}_n \}$ be a collection of ER rules, where `R` is the candidate rule, and $\texttt{R}_1, \ldots, \texttt{R}_n$ are rules already in the Rule Repository. If $U = D_1 \times D_2$, the goal of computing likely false negatives is to find a small set $\texttt{LFN(R)} \subseteq U \setminus links(\texttt{R})$, such that the links in `LFN(R)` are more likely to be labeled as positives by a human user than some arbitrary pairs in $U \setminus links(\texttt{R})$. Intuitively, `LFN(R)` contains pairs that are potentially useful for discovering new ER rules.

Computing `LFN(R)` is a challenging task because the search space of `LFN(R)` is extremely large. As illustrated in Figure 4.11, the coverage of $\mathbf{R}$ (indicated by the

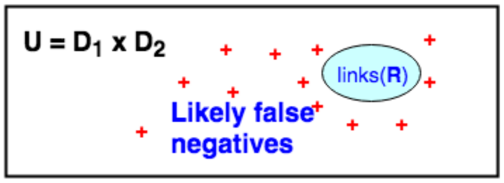


Figure 4.11: Search space for likely false negatives

ellipse) is typically very small compared to its complement.

When D_1 and D_2 are small, `LFN(R)` can be computed by sorting all pairs in U according to their similarity scores and returning the pairs with highest similarity scores. However, in realistic cases, the scale of the datasets makes the exhaustive examination impractical. In order to effectively and efficiently search for likely false negatives, we introduce a heuristic which we call Rule-Minus. The idea is to generate a set of rules that are generalizations of the current candidate rule `R`, in order to extend the space of potential matches beyond what is covered by the rules in **R** (but without an exhaustive search of U) .

We say a rule `S` is a *generalization* of a rule `R` if $links(\texttt{R}) \subseteq links(\texttt{S})$. A simple way to generalize a rule is by dropping matching predicates. More precisely, given `R` consisting of n matching predicates and *jump size* $s \geq 1$, let $\texttt{R}^{-}(s)$ be a rule obtained by dropping *at most* s predicates from `R` (assuming $n \geq s$). It is easy to see that $\texttt{R}^{-}(s)$ is a generalization of `R`. Additionally, if $\texttt{R}^{-}(s)$ has at least one blocking predicate then we refer to it as a Rule-Minus rule of `R`. As explained in Section 4.1, this is to ensure that we can efficiently evaluate Rule-Minus rules. We use $RM(\texttt{R}, s)$ to denote the set of all Rule-Minus rules that can be obtained from `R` using jump size s.

Example 4.6 Recall `Rule2` from Figure 4.5. Assuming that `T.lastname = C.lastname` is the only blocking predicate, the Rule-Minus rules of `Rule2` with jump size $s = 1$ are:

```
(1)  T.lastname = C.lastname AND firstNameMatch(T.firstname, C.firstname)

(2)  T.lastname = C.lastname AND sameHandle(T.screen_name, C.emailHandle).
```

However, the following rule will not form a valid Rule-Minus rule because it has no blocking predicate.

```
firstNameMatch(T.firstname, C.firstname)

                    AND sameHandle(T.screen_name, C.emailHandle).
```

$\square$

Let $\mathbf{L}$ denote the links covered by the candidate rule `R` along with all the rules in the repository. If $\mathtt{R}^- \in RM(\mathtt{R}, s)$, then $links(\mathtt{R}^-) \setminus \mathbf{L}$ represents, intuitively, a boundary between known (i.e., the links in $\mathbf{L}$) and unknown (i.e., arbitrary pairs in U). Our goal is to look for potential matches within the boundary set $links(\mathtt{R}^-) \setminus \mathbf{L}$. One may ask how likely it is that the set $links(\mathtt{R}^-) \setminus \mathbf{L}$ contains any matches. Recall that $\mathtt{R}^-$ and `R` differ in at most s predicates. Denote by `P` the set of predicates included in `R` but not in $\mathtt{R}^-$. Since any $l \in links(\mathtt{R}^-) \setminus \mathbf{L}$ satisfies all predicates in `R` barring the ones in `P`, l is far more likely to be a match than a pair chosen arbitrarily from U that may not satisfy a single predicate in `R`. Intuitively, $\mathtt{R}^-$ is obtained by a minor perturbation of `R` and a link resulting from the perturbed rule is more likely to be a match than an arbitrary pair from the much larger space U.

Given $\mathbf{R} = \{\mathtt{R}, \mathtt{R}_1, \ldots, \mathtt{R}_n\}$, we compute $\mathtt{LFN}(\mathbf{R})$ as follows: (1) generate the set

$RM(\mathtt{R}, s)$ of all possible Rule-Minus rules of R, (2) compute $|RM(\mathtt{R}, s)|$ sets of links, each obtained by applying a distinct Rule-Minus rule $\mathtt{R}^- \in RM(\mathtt{R}, s)$ on the input datasets, (3) for each set of links obtained in step (2), eliminate those links that are covered by **R**, (4) for each set of surviving links from step (3), select m links with highest similarity scores (where m is a fixed constant), and (5) output LFN(**R**) as the union of the sets of links obtained in step (4).

Note that in step (4), we return the m links with the highest similarity scores, for each one of the rules in $RM(\mathtt{R}, s)$. Alternatively, one may first take the union of all surviving links in step (3) and then return links with highest similarity scores. However, the intuition of the Rule-Minus heuristic is to explore different parts of the space outside of $links(\mathbf{R})$. Thus, taking the union first may destroy the diversity created by the multiple Rule-Minus rules; instead, we find top-m pairs in each of the individual sets, before applying the union. It is also interesting to consider the effect of varying the parameter s. It is easy to see that $RM(\mathtt{R}, s) \subseteq RM(\mathtt{R}, s')$ where $s \leq s'$ which in turn implies that:

$$\cup_{\mathtt{R}^- \in RM(\mathtt{R},s)} links(\mathtt{R}^-) \setminus \mathbf{L} \subseteq \cup_{\mathtt{R}^- \in RM(\mathtt{R},s')} links(\mathtt{R}^-) \setminus \mathbf{L}$$

which clearly illustrates that the space in which we look for likely false negatives increases as we increase the jump size. This can be an advantage if it leads to the discovery of more matches however it also implies that the Rule-Minus rules for s' are more general and may need more time to run. In Section 4.4, we describe experimental results that show the effectiveness of the Rule-Minus heuristic and the effect of varying jump size.

4.3.3 User Labeling Module

The system forwards each likely false positive and each likely false negative to the human user one at a time. The user is also provided the option of terminating the labeling process at any time (e.g., if the user's label budget has been reached). We also use specially developed heuristics to further reduce the number of label requests. While in general it may be necessary to label the whole set `LFP(R)`, in many cases (especially when `R` is not high precision), the first few labels may be enough to determine that $Prec_{\texttt{LFP(R)}}(\texttt{R}) < \tau$. The system counts the number of negative labels identified by the human and stops making further labeling requests if enough have been accumulated to ensure that the single-rule learning module will produce a different candidate ER rule in the next iteration (details provided in the full version of the paper).

4.3.4 Interaction with the Rule Repository

Our system accepts a candidate ER rule only if it satisfies the precision threshold τ. Such rules are then collected in the Rule Repository. Once no more high precision rules can be found, the system returns the disjunction of all the rules collected in the Rule Repository. By forming a disjunction out of all the high precision ER rules, the final ER algorithm achieves a higher recall than is possible with each individual rule, while still maintaining precision higher than τ. Not only does the Rule Repository act as a store for ER rules, it also performs the following crucial functions:

- Filter for labeling requests: Recall that, when computing `LFP(R)` and `LFN(R)`, in both cases a link results in a labeling request only if it is not covered by any rule in the Rule Repository. As explained in Section 4.3.2, in the case of `LFP(R)` this reduces the likelihood of including matches, while in the case of `LFN(R)` this is done to explore new links not covered by `R` or any of the already accepted rules.

- Ensures variety in ER rules: Once a high precision ER rule `R` is included into the Rule Repository, we remove all matches covered by `R` from the training data. This achieves two things. First, although `R` was a high precision rule with respect to the original training data, it is neither high precision nor does it maximize recall with respect to the modified training data; thus, in the next iteration, the single-rule learning module will likely produce a candidate distinct from `R`. Second, all ER rules added to the Rule Repository after `R` cover matches from the training data that are not covered by `R`. In turn, this ensures that the final disjunction returned by the system has high recall.

4.4 Experimental Evaluation

In this section, we provide an experimental evaluation of our system, which we call ERLEARN, and show that it is able to learn complex, high-precision, and interpretable rules at scale. We will also show that it outperforms a range of existing state-of-the-art learning methods including: an SVM-based pairwise ER classifier, a supervised learning method based on Markov Logic Networks (MLNs) [48], and the active learning method by Arasu et al [6].

Datasets. We consider two real-world matching scenarios.

EMP-TWITTER scenario. In this scenario, the task is to identify matches between a large collection of user profiles obtained from Twitter and an internal set of company employee records, denoted as EMP (the name of the company is omitted for anonymity purposes).While the EMP dataset has a variety of attributes, the ones that are relevant for matching are: `STATE`, `CITY`, `COUNTRY`, `EMAIL_ADDRS` (a list of emails including personal emails and work emails), and `Name` (a combination of `first`, `middle`, `last`). In addition, we also compute a derived attribute, `firstNameVars`, which is a list of pre-computed variations of the first name. From the Twitter profiles, the attributes that we use for matching are: `url`, `home` (a combination of `city`, `state`, and `country`), `name` (a combination of `first`, `last`, `middle`, and `full_name`), `gender`, and `occupation_mentions` (a combination of `category`, `date`, and `job_type`). There are many other attributes in the Twitter data, such as `friend_count` and `user_verified`, but they are not immediately relevant for the matching task. The EMP dataset contains 467,761 records, while the Twitter dataset that we use contains 50 million user profiles. We also note that each of the datasets is largely free of duplicates (i.e., each record denotes a unique person) and the goal is to link across the two datasets.

CRYSTAL scenario. In this scenario, we use a single input dataset containing records about companies, many of which are duplicates, and the task is to identify the pairs of records that are duplicates. The `Crystal` data contains several attributes relevant for matching: `country`, `company_id`, `industry`, `name`, `parent`, `source`, and `sub-industry`. The `Crystal` dataset contains 1,334,766 records, collected from multiple external data

Table 4.1: Statistics of datasets

	Size	**Predicates**
Emp-Twitter	50M$\times$470K	68
Crystal	1.3M$\times$1.3M	158

sources (including S&P Capital IQ).

In both scenarios, we considered a relatively large number of matching predicates, 68 (Emp-Twitter scenario) and 158 (Crystal scenario), which are larger than what is typically considered in previous approaches (e.g., [6]). These predicates are constructed, as explained in Section 4.1, by combining in various ways all the available matching functions and unary functions that are applicable on the relevant attributes. The statistics of our datasets is summarized in Table 4.1.

Methodology. When evaluating ERLEARN, we are interested in understanding: (1) the number of rules learned for each matching scenario, together with their precision (the precision of rule `R` is estimated by evaluating a randomly selected subset of 30 links in *links*(`R`)); (2) the overall recall, which is estimated by the total number of links, as discussed in Section 4.2.1; (3) the number of labeling requests that are needed to achieve both high recall and high precision; (4) the differences between ERLEARN and other approaches with respect to the above aspects; and (5) the impact of varying different parameter settings. To better understand the trade-off between labeling effort and recall, we introduce a metric, which we call *cost effectiveness*, to measure how effectively a learning algorithm converts user labelings into high-precision links. We define the cost effectiveness of a learning method X as

$$\alpha(X) = \frac{|links(X_{\texttt{Algo}})|}{\#\ labelings\ required\ by\ X\ to\ produce\ X_{\texttt{Algo}}},$$

where $X_{\texttt{Algo}}$ is the ER algorithm produced by X. Higher values of $\alpha(X)$ indicate better performance. In our experiments, all labelings were performed by the authors.

Parameters. Important tunable parameters include: the number of pre-labeled examples (*Prelabels*); jump size (*JS*) for Rule-Minus rules; precision threshold τ used both in the single-rule learning algorithm and in the estimation of precision of a candidate rule; the number of likely false positive examples that are produced in each iteration (parameter k, recall Section 4.3.2); the number of likely false negative examples to be labeled per Rule-Minus rule in each iteration (parameter m, recall Section 4.3.2). We set $k = 10$, $m = 2$, and $\tau = 90\%$ for all ERLEARN experiments.

Implementation. The learning system is implemented in Java, while the execution of the ER rules during active learning is done via HIL [42], running on a Hadoop cluster with 7 nodes (one master and six slaves), where each node is a 64-bit machine with 6 cores and hyper-threading technology.

4.4.1 Evaluation of ERLEARN: Hightlights

Table 4.2 provides an overview of the results of running ERLEARN on two matching scenarios. All ER rules generated by ERLEARN have precision higher than 90%, which justifies our conservative estimation method, discussed in Section 4.2.1. For the Emp-Twitter scenario, ERLEARN learned four (resp. seven) rules that give ∼680

Table 4.2: A summary of results - ERLEARN

ID	**Prelabels**	**JS**	**Iters.**	**Rules**	**Links**	**Labels**
ET-80-1	80	1	25	4	680	166
ET-30-1	30	1	26	4	683	172
ET-30-2	30	2	48	7	1088	430
`Crystal`	350	1	24	3	145,516	147

(ID prefix ET denotes Emp-Twitter)

(resp. 1088) links when JS=1 (resp. JS=2), and ERLEARN achieved that by asking for a reasonable number of user labels (∼170 and 430). For the Crystal scenario, ERLEARN learned three rules that give about 145K links by asking for only 147 user labels. For the two scenarios, the number of links produced by ERLEARN differ by orders of magnitude. This is because the two matching tasks are very different in nature. The Crystal scenario is a deduplication scenario with a large number of duplicates in its input dataset, which inherently results in a large number of links. In contrast, in the Emp-Twitter case, the matching ratio is bounded by the size of the Emp employee data and also by the fact that many Emp employees may not tweet. Moreover, Twitter data is very sparse and for many Twitter profiles there is limited information to reveal the identity of the person. Intuitively, this is a more challenging ER task.

We next give a detailed comparison of ERLEARN with other state-of-the-art methods on the larger of the two scenarios (Emp-Twitter) since it is more challenging. Figure 4.12 shows two rules (one from each scenario) learned by ERLEARN. The rule `R1` is a complex rule that consists of five predicates comparing first name (and variations thereof), last name (taking into account its frequency in census data), and geographic locations. This rule is easy to understand, looks at richly structured data (`firstNameVars`

```
match Emp i, Twitter t by R1:
 firstNameMatch(i.Name.firstNameVars,t.name.first)
 AND i.Name.last = t.name.last
 AND lastNameFrequencyFilter(t.name.last, 60)
 AND upperCase(i.CITY) = upperCase(t.home.city)
 AND countryIsInUSA(i.COUNTRY)

match Crystal c1, Crystal c2 by R2:
 c1.country = c2.country
 AND c1.industry = c2.industry
 AND c1.subIndustry = c2.subIndustry
 AND Jaccard_Similarity(c1.name, c2.name, 90)
```

Figure 4.12: Two real ER rules learned by ERLEARN

is a list), and relies on multiple attributes to determine links. The ability to learn the "correct" combination of predicates separates ERLEARN from other approaches. For instance, consider the filter `lastNameFrequencyFilter(x,`θ`)` which returns true only if `x` is not in the top θ% most popular last names. If rule `R1` did not contain this filter, then the modified rule would likely return a large number of false positives since many people living in the same city have similar first and last names. One of the approaches we compare against, based on Markov logic networks (MLN) [61] (see Section 4.4.2), produces such a rule without the filter (see MLN-learned rules in Figure 4.13 in Section 4.4.2). Furthermore, even if the frequency filter is included, a crucial challenge is to learn the appropriate threshold. As an example, another approach we compare against (ALGPR [6], see Section 4.4.2) learns a version of `R1` with $\theta = 85\%$. However, when compared to our `R1`'s $\theta = 60\%$, this results in a more aggressive filtering than necessary, leading to significantly lower recall. Similar observations apply to the CRYSTAL scenario.

4.4.2 Comparison with Existing Methods

We compare ERLEARN with the following approaches:

- SVM: a statistical classifier that classifies pairs of records using a support vector machine.
- MLN: a first-order probabilistic framework using Markov Logic Networks [61], with known learning methods [47, 48].
- ALGPR: a rule-based active learning algorithm for ER [6].

Table 4.3 reports all results on EMP-TWITTER scenario where * denotes supervised learning methods that require fully labeled training data, α denotes cost effectiveness, bold fonts denote links obtained at high precision ($\geq 90\%$) and X-Y denotes an approach where we learned rules using method X and performed ER using inference method Y.

SVM. A number of prior works on ER utilized various statistical classifiers [37, 62] and SVMs are one of the most accurate classifiers available. Our implementation uses the LIBSVM package [53], and the parameter settings used in our implementation is shown below (a detailed explanation of the meaning of each parameter can be found at [53]).

```
java svm_train -b 1 s 0 -c 100 -6 0 -e 0.0001 train.data
```

To train the SVM, which is a supervised classifier, we provided a set of 430 labeled pairs collected using ERLEARN's active learning approach. Performing ER using the out-of-the-box learned SVM would require materializing the underlying feature vectors for all pairs in the cross product, which is infeasible given EMP-TWITTER scenario's

Table 4.3: A comparison of all approaches on EMP-TWITTER scenario.

Methods		γ	Labels	Links	Precision	α
SVM*		0 0.8 0.95	430	10,849 564 **21**	$< 3.3\%$ 6.7% 95.2%	 0.049
MLN-PSL*		0 0.9	430	21,575 **84**	$< 3.3\%$ 90%	 0.195
ALGPR		n/a	159	**181**	93.3%	1.14
ERLEARN-HIL	JS=1	n/a	172	**683**	93.3%	3.97
	JS=2	n/a	430	**1088**	93.3%	2.533
ERLEARN-PSL		0 0.001 0.01	430	1,169 **1,090** 297	86.7% 90% 100%	2.534

data size. Instead, we apply a blocking function (`twitter.lastname = Emp.lastname`) to eliminate obvious non-matches and then apply the SVM to pairs within the same block. Table 4.3 shows that SVM's precision is extremely poor even though it produces 1000s of links. This is a direct result of the loss function used for training LIBSVM's SVM which, like most statistical classifiers, attempts to maximize overall accuracy of predictions. However, in the ER setting, where non-matches far outnumber matches, overall accuracy of classification may not be the best indicator (it is far more important to be correct on the true matches portion of the space). To improve the SVM's precision, we took the score ($\in [0, 1]$) predicted for each pair and called it a link only when the score exceeded a certain threshold γ. Table 4.3 shows that SVM produces only 21 links at high precision (achieved at very high $\gamma = 0.95$) and is the least cost effective approach among all methods tried.

MLN. MLNs are a sophisticated class of statistical models that can make joint decisions and have been used to perform ER [65]. An MLN model consists of weighted rules that can be either given or learned using various learning algorithms [47, 48]. While MLNs can also perform inference using the set of rules, *Probabilistic Soft Logic* (PSL), which is another first-order statistical inference framework, is more efficient and has been demonstrated to produce more accurate results [11][3]. For our implementation, we first learn MLN rules by learning structured motifs [48, 54], followed by learning weights for the rules using PSL [59], which also performs the actual inference of the matches. We use the same training data that we used for training SVMs, and the same blocking function to scale inference in the EMP-TWITTER scenario. MLN-PSL learned three rules, and two of them are shown in Figure 4.13.

```
match Emp i, Twitter t by R1:
 firstNameMatch(i.Name.firstNameVars,t.name.first)
 AND i.Name.last = t.name.last
 AND upperCase(i.CITY) = upperCase(t.home.city)

match Emp i, Twitter t by R2:
 firstNameMatch(i.Name.firstNameVars,t.name.first)
 AND i.Name.last = t.name.last
 AND i.emailHandler = t.screen_name
 AND upperCase(i.CITY) = upperCase(t.home.city)
```

Figure 4.13: Two real ER rules learned by MLN

As in the case of SVM, since MLNs are not constrained by a high-precision threshold during training, they return a large number of links but at very poor precision (see Table 4.3). If we use a threshold and select pairs with high scores, then precision can be improved but that leaves us with only 84 links. While recall is low, PSL scores are reliable indicators of true matches. More precisely, scores for true matches exceeded

[3]Note that PSL does not automatically learn rules.

0.9 which was much greater than scores for other links. Despite this, MLN-PSL's poor quality rules make it a not so ideal choice for ER.

ALGPR. In contrast to the two previous supervised learning approaches, The work that is most similar to ours is by Arasu et al [6], which introduced an active learning algorithm ALGPR whose goal is also to maximize recall under a precision constraint. In [6], textual similarity measures between two records are used to decide if they are a match or not. Thus, given a collection of d similarity measures, they consider a d-dimensional feature space where each dimension represents a similarity measure. More concretely, in this d-dimensional space, each point $(\theta_1, \ldots, \theta_d)$, where $\theta_i \in [0, 1]$, represents an ER rule $sim_1(r, s) \geq \theta_1 \wedge \cdots \wedge sim_d(r, s) \geq \theta_d$, where (r, s) is a pair of records. The goal is to learn the optimal thresholds θ_i's such that the resulting ER rule achieves high recall while still satisfying a predefined precision constraint. Like us, Arasu et al also discretize the space of possible values of θ_i's by using fixed increments (e.g., 0.1). To learn the thresholds, ALGPR starts with an ER rule $\mathtt{R}_{(1,\ldots,1)}$ which is the most conservative rule (high-precision, low recall) with all θ_i's set to 1. The algorithm then searches for the optimal thresholds essentially through a binary search along each dimension, in an attempt to increase the recall while keeping precision above a threshold τ. For each search point along each dimension, the algorithm has to evaluate the precision of the corresponding rule by issuing labeling requests to a user. Arasu et al then extend their algorithm to a simple greedy algorithm that is able to generate an s-term DNF formula (see [6] for more details). In our evaluation, we implemented the extended version of ALGPR that generates an s-term DNF formula [4]. For ALGPR, in our experiment, we set the precision threshold at 90%, and use 0.1 as the increment when searching for the values for θ_i's.

Figure 4.14 shows the two rules learned by ALGPR. Table 4.3 shows that, ALGPR produces high precision rules while requiring fewer labels. ALGPR is about 23.2 (resp. 5.8) times more cost effective than SVM (resp. MLN-PSL) on EMP-TWITTER scenario. However, only the first ALGPR rule provides significant recall. This implies that, not looking for false negatives while learning (*a la* Rule-Minus heuristic) may result in sub-optimal recall.

```
match Emp i, Twitter t by R3:
 upperCase(i.CITY) = upperCase(t.home.city)
 AND i.emailHandler = t.screen_name
 AND i.Name.last = t.name.last
 AND lastNameFrequencyFilter(i.last, 85)

match Emp i, Twitter t by R4:
 upperCase(i.CITY) = upperCase(t.home.city)
 AND i.Name.last = t.name.last
 AND lastNameFrequencyFilter(i.last, 80)
 AND i.emailHandler = t.screen_name
 AND upperCase(i.STATE) = upperCase(t.home.state)
```

Figure 4.14: Two real ER rules learned by ALGPR

ERLEARN. Among all the approaches, ERLEARN provides the best trade-off between labeling effort and recall since Table 4.3 shows that all three results involving ERLEARN produce much higher cost effectiveness than SVM, MLN-PSL and ALGPR. For ERLEARN-HIL (ERLEARN using HIL for evaluation of rules), we report results with 2 settings of jump size. While JS=1 produces the best cost effectiveness, JS=2

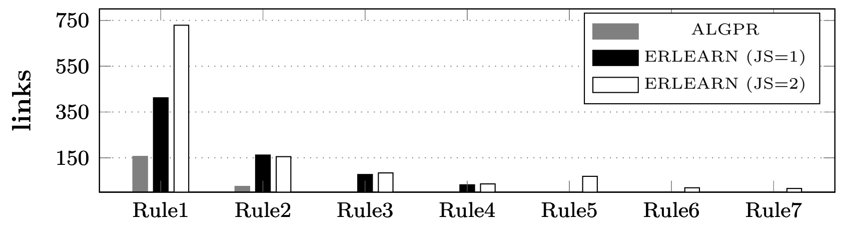
[4]Our implementation also fixes a minor mistake which the authors shared with us in private communication.

produces significantly better recall (405 additional links) although this comes at the cost of increased labeling effort (258 more). This shows that more general Rule-Minus rules lead to more false negatives. Note that it does not make sense to increase JS beyond 2 because more often than not, after dropping 2 predicates the only predicate remaining in the Rule-Minus rule is the blocking predicate. Besides executing the rules using HIL, we also report results with ERLEARN-PSL wherein we used PSL as an alternative way of inferring matches given ERLEARN's rules (learned with JS=2). Note that PSL inference automatically learns weights for the rules, to possibly differentiate among them. Table 4.3 shows that ERLEARN-PSL and ERLEARN-HIL give similar number of links at high precision. This implies that the extra step of learning weights when rules are already of high precision does not provide extra benefit. It also shows that ERLEARN's rules are robust enough to be used with deterministic or probabilistic evaluation engines.

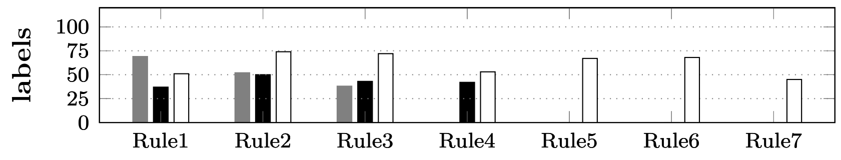
In summary, ERLEARN provides not only the best cost effectiveness among all approaches but also the highest recall (at high precision), with ALGPR being a distant second. We take a closer look at these two approaches next.

4.4.3 Quality of Rules Learned by ERLEARN

Figure 4.15(a) shows the number of new links given by each ER rule in the two methods (the bar for `Rule` i, where $i = \{2, \ldots, 7\}$, counts links produced by `Rule` i that are not already covered by `Rule` 1 through $i - 1$). Figure 4.15(b) shows a breakdown of the labeling requests (required to learn each rule). For ALGPR, the first rule yields 156 links by labeling 69 examples. In contrast, the first rule of ERLEARN yields significantly



(a) Number of new links produced by each rule



(b) Labeling effort

Figure 4.15: A detailed comparison of ERLEARN and ALGPR

more links (412 for JS=1 or 729 for JS=2) than ALGPR's first rule by requesting even fewer labels. The second rule of ALGPR gives 25 new links by labeling 52 examples, while ERLEARN's second rule gives 162 (resp. 155) new links by requesting 50 (resp. 74) user labels when JS=1 (resp. JS=2). We noticed that ALGPR required the user to label 75% more examples to get a second ER rule, but the new rule yields about 16% more links. This result is comparable with the observations in [6], where the authors reported that the second rule generated by their system improves the overall recall by about 2%-5%, while asking the user to label about 50% more examples. In contrast, ERLEARN's subsequent rules give much higher recall. Specifically, the 2^{nd} (resp. 3^{rd}, 4^{th}) rule generated by ERLEARN (JS=1) gives 39% (resp. 13.4%, 4.9%) new links with respect to all earlier rules combined, while ALGPR cannot find any more rules after generating two rules. When the jump size is set to two, ERLEARN can find three more rules, and the overall recall was significantly increased.

Figure 4.15(b) gives a detailed look at the number of labeling requests required by the two methods. We can see that the labeling effort in ERLEARN stays at about the same (relatively small) level for each of the four ER rules in the JS=1 case while exhibiting a graceful degradation in terms of links obtained (similarly, for the first five rules in the JS=2 case). Compared to ALGPR in terms of cost effectiveness, ERLEARN with jump size one (resp. two) can find 2.8 (resp. 1.4) more links than ALGPR by requesting one user labeling. We noticed that, in the JS=2 case, ERLEARN asked a fair amount of user labelings (113 requests) so as to produce `Rule6` and `Rule7`, which are low-recall (15 links combined). In our experiments, we pushed ERLEARN to its limit, so it terminated when single-rule learning module does not produce any rule that satisfies the precision threshold. ERLEARN can also report the number of links covered by the newly accepted high-precision rule, and the user may use that information to terminate the system if the user believes no more useful rules (with significant recall) can be found. Stopping the system automatically at an optimal point is a subject for future system optimization.

4.4.4 Rule-Minus Heuristic and Other Aspects

Effectiveness of Rule-Minus. The superiority of ERLEARN over other solutions is, in part, due to the Rule-Minus heuristic, which is quite effective in finding false negative examples. Table 4.4 shows the number of likely false negatives labeled by the user and how many of them turned out to be matches. The Rule-Minus heuristic is very productive because, in both scenarios, a significant portion (78% when JS=1 and 42.8% when JS=2) of likely false negatives selected by the Rule-Minus heuristic were identified as true matches. As a result, the high conversion rate leads to new rules thus improving the overall recall. To further show the impact of the Rule-Minus heuristic, we reran ET-30-1 (Table 4.2) with the Rule-Minus heuristic disabled. This experiment resulted in one rule with 162 links (at precision $> 90\%$) in seven iterations which is significantly worse than the recall (i.e., 683 links) obtained when the Rule-Minus heuristic was enabled.

Table 4.4: # False negatives found by Rule-Minus rules

ID	**Likely false negatives**	**Matches**
ET-30-1	106	82
ET-30-2	238	102
`Crystal`	94	74

Runtime efficiency. Each iteration of ERLEARN can be divided into three phases: single-rule learning, rule evaluation, and user interaction. Among the three, the rule evaluation and user interaction phases constitute more than 95% of the overall runtime. The Rule-Minus heuristic has a substantial impact on both of these phases. Specifically, if the candidate rule consists of n predicates (typically, $n \in [3, 5]$), ERLEARN would execute 1 candidate and at most $\binom{n}{\texttt{JS}}$ Rule-Minus rules, and the user would have to label at most k likely false positive and $m\binom{n}{\texttt{JS}}$ likely false negative examples, where JS denotes the jump size. Due to our efforts to reduce the number of required user labels (Section 4.3.3), the impact of increasing JS is less pronounced than the above analysis suggests. Table 4.2 shows that on average ERLEARN incurs 6-7 labels per iteration with $\texttt{JS} = \texttt{1}$ (in both scenarios) which increases to 8.9 at $\texttt{JS} = 2$. The time required to perform these labelings depends on the human user as well as on the scenario; in our experiments we spent one minute per label on average, which leads to about 6-9 minutes of user time per iteration. In contrast, since HIL needs to invoke MapReduce jobs to evaluate each rule (recall that, EMP-TWITTER scenario consists of more than 10^{13} pairs), the impact of JS on rule evaluation time is more clearly evident. Concretely, to evaluate the rules ERLEARN required about 5 to 8 minutes (resp. 35 to 45 minutes) per iteration on average when JS=1 (resp. JS=2) in the EMP-TWITTER scenario. Therefore, while increasing the jump size results in better recall, it comes at the cost of increased runtime required to evaluate rules. One immediate way to reduce the rule evaluation time is to add more nodes to the cluster and/or to deploy the system on a more efficient platform such as Apache Spark. More generally, automatically determining a jump size that strikes the optimal balance between runtime efficiency and quality of learning, given the available resources, is a direction for future work.

Impact of prelabels. As shown by the two experiments ET-80-1 and ET-30-1 (Table 4.2), the initial number of pre-labeled examples (i.e., 80 vs. 30) does not have much impact on the final outcome. As discussed above, this is because the Rule-Minus heuristic can find enough additional examples during its active learning iterations to compensate for the lack of pre-labeled examples in the second case. In fact, we expect the system to produce similar results by using even less (or even an empty set of) pre-labeled examples.

Label budget. Figure 4.16 shows that ERLEARN produces better recall than ALGPR at all label budgets (in each plot, each step denotes the labels required to learn the next rule). If we have a large budget, e.g., 200, then ERLEARN can produce 683 links (or ∼900 when JS=2) while ALGPR produces less than 200 links. ERLEARN outperforms ALGPR even in the case of a tighter label budget, e.g., 100, with the former producing 574 or 729 links (depending on JS) vs. the latter's 156 links.

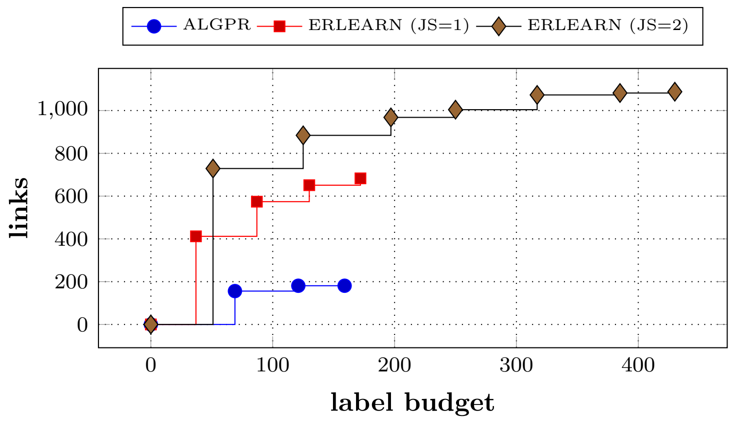


Figure 4.16: Trade-off between label budget and links.

4.5 Related Work

For a general overview of various ER approaches and algorithms, we refer the interested reader to excellent surveys and tutorials [31, 35, 36]. In what follows, we describe work that is closely related to our approach, which is focused on learning of the ER algorithms in a target language.

While we propose an active learning approach, other approaches to ER learning include unsupervised and supervised learning methods [36], the latter usually leading to more accurate results. To apply supervised learning one requires labeled training data. For ER, this usually implies labeling a large number of examples since the non-matches far outnumber matches and a substantial amount of human effort needs to be invested up front so that the training data consists of more than just a handful of matches. Markov Logic Networks (MLN) [61] represent a sophisticated class of supervised learning models that aim to unify logic and uncertainty. While ERLEARN aims to learn monotone DNF formulas, which is tractable in a precise sense (see [5]), learning general MLN rules is intractable. However, efficient heuristics are available; in particular, LSM [48] is capable of learning longer MLN rules than other approaches (e.g., [47]), which is why we chose it for our MLN comparison. Active learning for MLNs has been attempted in [33]; however, that approach does not use the full power of MLNs and, moreover, it expects a user to manually write the rule every time a misclassification happens.

Supervised learning attempts to minimize *0-1 misclassification loss* or the number of incorrect predictions. However, due to the extreme imbalance of non-matches and matches, a trivial classifier that always predicts a pair to be a non-match will incur a very low misclassification loss, but would be useless for ER purposes. The method in [12] learns a high precision classifier by making multiple calls to a misclassification loss minimizing learner. However, the cumulative labeling complexity may far exceed the labeling effort incurred by a direct approach (e.g., ERLEARN) that avoids calls to an intermediary, especially for large data.

Active learning is a systematic way to learn accurate models while requiring far fewer labels than supervised learning, which is why it has been used with some degree of success for ER [6,62,66]. A core component in active learning is the strategy for choosing examples to request labels for. Previous approaches have used *uncertainty* [62,66] and *non-redundancy* [13,71]. [62] measures uncertainty of an example by counting how many statistical models in a committee disagree on its label. On the other hand, [71] uses clustering algorithms to determine examples that are far apart in the feature space to forward to the labeler. [13] goes a step further to ensure that the examples are representative of a significant portion of the data. ALGPR [6] chooses examples by randomly sampling from the links returned by a rule. In contrast, our strategy is guided by both likely false positives and likely false negatives which is arguably a better fit with the objective of maximizing recall while maintaining high precision.

The Emp-Twitter scenario bears a passing resemblance to user profile matching [57, 72] whose goal is to match users across social networks. Due to the difficulty of gaining access to social network data, most user profile matching experiments are performed on only a subset of the network. As [37] shows, the characteristics of these subsets are very different from the characteristics of the real world data and their ER results no longer apply to real world social networks. In contrast, we performed our experiments on the entirety of Emp-Twitter scenario and Crystal scenario, both of which contain real world data. In other works, crowdsourcing has also been used to identify record pairs that represent the same entity [55, 70]. However, besides not producing an ER algorithm that can be interpreted and maintained, in the big data setting there may be too many tasks issued to the crowd thus drastically increasing the costs associated with such approaches.

4.6 Conclusions

We introduced a new active learning system, ERLEARN, that can scalably learn high-quality ER algorithms on big data. The main feature of ERLEARN is the ability to learn multiple high-quality ER rules that are significantly different from each other by actively exploring both likely false positives and likely false negatives. In a comprehensive experimental evaluation, on two real-world big data matching scenarios, we showed the scalability and effectiveness of ERLEARN. When compared to previous methods, ERLEARN gives the best trade-off between quality of results and labeling effort. In the future, we intend to evaluate ERLEARN on more datasets, study the effect of varying parameters, and design improved convergence criteria.

Chapter 5

Concluding Remarks

In this dissertation, we described our work on developing example-driven approaches to designing high-level schema mapping specifications and high-level rule-based entity resolution algorithms. Here, we briefly summarize the main contributions made and list a few possible directions for future work.

A Summary of Contributions. We embarked on research that extends, refines, and investigates two example-driven schema-mapping discovery frameworks, namely, the Gottlob-Senellart framework and the learning framework. In particular, for the Gottlob-Senellart framework, we investigated the derivation of an optimal schema mapping from a set of data examples and focused on the approximation properties of this optimization problem. We considered several different sublanguages of the language of GLAV schema mappings and delineated the boundary as regards good approximation properties. We established negative results about the existence of approximation algorithms for GAV and GLAV schema mappings, but also showed that the class of SH-LAV schema mappings is the largest subclass of GLAV mappings for which we can obtain a positive approximation result. In addition to the complexity-theoretic study, we also implemented the proposed $\log(n)$-approximation algorithm for SH-LAV schema mapping and evaluated it on a real-world mapping scenario. It is the first concrete algorithm, to the best of our knowledge, that derives schema mappings under the Gottlob-Senellart framework.

Regarding the learning framework, we proposed the GAVLearn algorithm that is a PAC learning algorithm for the class of GAV schema mappings. We also showed that if GAVLearn is provided with an oracle for NP then GAVLearn is an efficient PAC learning algorithm. We implemented the GAVLearn algorithm and evaluated it using mapping scenarios created by iBench. As a byproduct of the experimental evaluation, we also introduced a new measure, namely, the F-score measure, to evaluate the quality of a schema mapping with respect to a set of data examples. By a comparison with two existing methods, we showed that the GAVLearn algorithm was able to produce schema mappings that were better (in terms of F-scores) than the mappings produced by the other two methods with respect to mapping scenarios created by iBench.

Regarding learning rule-based ER algorithms, we presented a new active learning system, namely, ERLEARN, that learns high-quality rule-based entity resolution algorithms at scale. The ERLEARN system actively searches for two types of data examples: false positives and false negatives. The former are used to improve the precision of a candidate rule. The latter are used to learn more high-precision rules that are different from previously learned high-precision rules, which in turn would improve the overall recall of the learned ER algorithm. We experimentally compared ERLEARN with existing methods (including a state-of-the-art active learning algorithm and a number of supervised learning methods) on two real-world big data matching scenarios. The experimental evaluation showed that ERLEARN gave the best trade-off between labeling efforts and quality of results.

Open Questions and Future Work. We designed and implemented the $\log(n)$-approximation algorithm for SH-LAV schema mappings, under the Gottlob-Senellart framework, with respect to the SH-LAV$^{=}$-repair language. Two important questions that remain to answer are the existences of polynomial time $\log(n)$-approximation algorithms for computing near-optimal LAV$^{=,\neq}$ and SH-LAV$^{=,\neq}$ schema mappings. As regards the learning framework, the learnability of the class of LAV schema mappings was left open, likewise for the class of GLAV schema mappings. Pursuing the answers to these questions is our future work.

Although the fitting framework was not investigated in this dissertation, there is still room for extending the fitting framework. Note that the fitting framework focuses on the fitting decision problem, namely, determining whether there is a fitting GLAV schema mapping for a finite set of data examples. It follows that if the answer is no, the current fitting GLAV generation algorithm would simple terminate and return "none exists". An important open question is that: is there a way to modify the provided set of data examples such that the resulting set of data examples admits a fitting GLAV schema mapping? This problem can be cast as an optimization problem, in which the goal is to minimize the “modifications” needed.

Currently, the ERLEARN system runs on command line. The next step is to built a graphic user interface upon the system. There are also several ways to improve the user experience, such as (1) developing an improved convergence criteria. For instance, the system may automatically terminate when the recall gain of a newly accepted high-precision rule is low; (2) developing techniques that further reduce the user labels. For instance, before asking for user labels, the system may first automatically reject pairs that are obvious non-matches (e.g., a Twitter profile and a customer record that have different country values). Moreover, we also plan to execute ERLEARN in a mapping scenario, in which ground truth is available, so that we can test the true recall of the ER algorithms learned by ERLEARN.

Bibliography

[1] Bogdan Alexe, Wang Chiew Tan, and Yannis Velegrakis. STBenchmark: Towards a Benchmark for Mapping Systems. *PVLDB*, 1(1):230–244, 2008.

[2] Bogdan Alexe, Wang-Chiew Tan, and Yannis Velegrakis. Stbenchmark: Towards a benchmark for mapping systems. *Proc. VLDB Endow.*, 1(1):230–244, August 2008.

[3] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Designing and refining schema mappings via data examples. In *SIGMOD*, pages 133–144, 2011.

[4] Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, April 2006.

[5] Dana Angluin. Queries and Concept Learning. *Machine Learning*, pages 319–342, 1988.

[6] Arvind Arasu, Michaela Götz, and Raghav Kaushik. On Active Learning of Record Matching Packages. In *SIGMOD*, 2010.

[7] Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with con-

straints using dedupalog. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 952–963, Washington, DC, USA, 2009. IEEE Computer Society.

[8] Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. The ibench integration metadata generator. *Proc. VLDB Endow.*, 9(3):108–119, November 2015.

[9] Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. The ibench integration metadata generator. *Proc. VLDB Endow.*, 9(3):108–119, November 2015.

[10] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The dl-lite family and relations. *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH (JAIR)*, 36:1–69, 2009.

[11] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *CoRR*, abs/1505.04406, 2015.

[12] Kedar Bellare, Suresh Iyengar, Aditya Parameswaran, and Vibhor Rastogi. Active Sampling for Entity Matching. In *KDD*, 2012.

[13] G. Dal Bianco, R. Galante, M. A. Gonsalves, S. Canuto, and C. A. Heuser. A practical and effective sampling selection strategy for large scale deduplication. *IEEE TKDE*, 2015.

[14] Mikhail Bilenko and Raymond Mooney. Adaptive Duplicate Detection Using Learnable String Similarity Measures. In *KDD*, pages 39–48, 2003.

[15] Alselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Inf. Process. Lett.*, 24(6):377–380, April 1987.

[16] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *VLDB*, pages 1267–1270, 2005.

[17] Angela Bonifati, Radu Ciucanu, and Slawek Staworko. Interactive join query inference with jim. *Proc. VLDB Endow.*, 7(13):1541–1544, August 2014.

[18] Angela Bonifati, Radu Ciucanu, and Slawek Staworko. Learning join queries from user examples. *ACM Trans. Database Syst.*, 40(4):24:1–24:38, January 2016.

[19] Douglas Burdick, Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. A declarative framework for linking entities. *ACM Trans. Database Syst.*, 41(3):17:1–17:38, July 2016.

[20] Balder ten Cate, Víctor Dalmau, and Phokion G. Kolaitis. Learning schema mappings. In *Proceedings of ICDT*, pages 182–195, New York, NY, USA, 2012. ACM.

[21] Balder ten Cate, Víctor Dalmau, and Phokion G. Kolaitis. Learning schema mappings. *ACM Trans. Database Syst.*, 38(4):28:1–28:31, December 2013.

[22] Balder ten Cate, Richard Halpert, and Phokion Kolaitis. Exchange-repairs: Managing inconsistency in data exchange. *Journal of Data Semantics*, 5(2):77–97, 2016.

[23] Balder ten Cate and Phokion G. Kolaitis. Schema mappings: A case of logical dynamics in database theory. In Alexandru Baltag and Sonja Smets, editors, *Johan Van Benthem on Logic and Information Dynamics*, chapter 3. Springer Publishing Company, Incorporated, 2014.

[24] Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang Chiew Tan. Approximation algorithms for schema-mapping discovery from data examples. In *Proceedings of the 9th Alberto Mendelzon International Workshop on Foundations of Data Management, Lima, Peru, May 6 - 8, 2015.*, 2015.

[25] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[26] Zhaoqi Chen, Dmitri V. Kalashnikov, and Sharad Mehrotra. Exploiting context analysis for combining multiple entity resolution systems. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 207–218, New York, NY, USA, 2009. ACM.

[27] V. Chvtal. A greedy heuristic for the set covering problem. *Math. Oper. Res.*, 4:233–235, 1979.

[28] Munir Cochinwala, Verghese Kurien, Gail Lalk, and Dennis Shasha. Efficient data reconciliation. *Inf. Sci.*, 137(1-4):1–15, September 2001.

[29] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.

[30] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing Schema Mappings: Second-order Dependencies to the Rescue. *ACM Transactions on Database Systems (TODS)*, 30(4):994–1055, 2005.

[31] Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Morgan & Claypool Publishers, 2012.

[32] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[33] J. Fisher, P. Christen, and Q. Wang. Active learning based entity resolution using markov logic. In *PAKDD*, 2016.

[34] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. Declaratively cleaning your data using ajax. In *In Journees Bases de Donnees*, 2000.

[35] Venkatesh Ganti and Anish Das Sarma. *Data Cleaning: A Practical Perspective*. Morgan & Claypool Publishers, 2013.

[36] Lise Getoor and Ashwin Machanavajjhala. Entity Resolution for Big Data. In *KDD*, 2013.

[37] O. Goga, P. Loiseau, R. Sommer, R. Teixeira, and K. P. Gummadi. On the reliability of profile matching across large online social networks. In *KDD*, 2015.

[38] Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. *JACM*, 57(2), 2010.

[39] Rahul Gupta and Sunita Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proc. VLDB Endow.*, 2(1):289–300, August 2009.

[40] Wolfang Gutjahr, Emo Welzl, and Gerhart Woeginger. Polynomial graph-colorings. *Discrete Appl. Math.*, 35:29–46, 1992.

[41] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *ACM SIGMOD*, pages 805–810, 2005.

[42] Mauricio Hernández, Georgia Koutrika, Rajasekar Krishnamurthy, Lucian Popa, and Ryan Wisnesky. HIL: A High-level Scripting Language for Entity Integration. In *EDBT*, pages 549–560, 2013.

[43] W. Iba and P. Langley. Induction of one-level decision trees. In *ICML*, 1992.

[44] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of STOC*, pages 38–49, New York, NY, USA, 1973. ACM.

[45] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, 1972.

[46] M. Kearns and L. G. Valiant. Crytographic limitations on learning boolean formulae and finite automata. In *Symposium on Theory of Computing*, 1989.

[47] Stanley Kok and Pedro Domingos. Learning markov logic network structure via hypergraph lifting. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 505–512, New York, NY, USA, 2009. ACM.

[48] Stanley Kok and Pedro Domingos. Learning markov logic networks using structural motifs. In *ICML*, 2010.

[49] KRDB. Obda mappings from imdb to ontology, 2013.

[50] KRDB. ontop project, 2013.

[51] KRDB. Sql script to generate the schema only imdb database, 2013.

[52] Hao Li, Chee-Yong Chan, and David Maier. Query from examples: An iterative, data-driven approach to query construction. *Proc. VLDB Endow.*, 8(13):2158–2169, September 2015.

[53] https://www.csie.ntu.edu.tw/˜cjlin/libsvm/, 2016.

[54] https://alchemy.cs.washington.edu/code/, 2010.

[55] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered Sorts and Joins. *PVLDB*, 2011.

[56] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema Mapping as Query Discovery. In *International Conference on Very Large Data Bases (VLDB)*, pages 77–88, 2000.

[57] M. Motoyama and G. Varghese. I seek you: Searching and matching individuals in social networks. In *Workshop on Web Information and Data Management*, 2009.

[58] Christos Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of STOC*, pages 229–234, New York, NY, USA, 1988. ACM.

[59] https://github.com/linqs/psl, 2016.

[60] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 475–484, New York, NY, USA, 1997. ACM.

[61] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning Journal*, 2006.

[62] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive Deduplication Using Active Learning. In *KDD*, 2002.

[63] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 1990.

[64] Yanyan Shen, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. Discovering queries based on example tuples. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 493–504, New York, NY, USA, 2014. ACM.

[65] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *ICDM*, 2006.

[66] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning Object Identification Rules for Information Integration. *Information Systems*, pages 607–633, 2001.

[67] Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. Query by output. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 535–548, New York, NY, USA, 2009. ACM.

[68] Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. Query reverse engineering. *The VLDB Journal*, 23(5):721–746, October 2014.

[69] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.

[70] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, pages 1483–1494, 2012.

[71] Q. Wang, D. Vatsalan, and P. Christen. Efficient interactive training selection for large-scale entity resolution. In *PAKDD*, 2015.

[72] G. You, S. Hwang, Z. Nie, and J. Wen. SocialSearch: Enhancing Entity Search with Social Network Matching. In *EDBT*, 2011.

[73] Meihui Zhang, Hazem Elmeleegy, Cecilia M. Procopiuc, and Divesh Srivastava. Reverse engineering complex join queries. In *Proceedings of the 2013 ACM SIGMOD*

International Conference on Management of Data, SIGMOD '13, pages 809–820, New York, NY, USA, 2013. ACM.

Index