Active Learning of GAV Schema Mappings

Balder ten Cate
Google Inc.
balder.tencate@gmail.com

Phokion G. Kolaitis
UCSC & IBM Research – Almaden
kolaitis@cs.ucsc.edu

Kun Qian
IBM Research – Almaden
qian.kun@ibm.com

Wang-Chiew Tan
Recruit Institute of Technology Inc.
wangchiew@recruit.ai

ABSTRACT

Schema mappings are syntactic specifications of the relationship between two database schemas, typically called the source schema and the target schema. They have been used extensively in formalizing and analyzing data inter-operability tasks, especially data exchange and data integration. There is a growing body of research on deriving schema mappings from data examples, that is, pairs of source and target instances that depict the behavior of the unknown schema mapping. One of the approaches used in this endeavor casts the derivation of a schema mapping from data examples as a learning problem. Earlier work has shown that GAV mappings (global-as-view schema mappings) are learnable in Angluin's model of exact learning with membership queries and equivalence queries. Here, we validate the practical applicability of this theoretical result by designing and implementing an active learning algorithm, called GAV-Learn that derives a syntactic specification of a GAV mapping from a given set of data examples and from a "black-box" implementation. We analyze the properties of GAV-Learn and, among other results, we show that it produces a GAV mapping that has minimal size and is a good approximation of the unknown GAV mapping. Furthermore, we carry out a detailed experimental evaluation that demonstrates the effectiveness of GAV-Learn along different metrics. In particular, we compare GAV-Learn with two earlier approaches for deriving GAV mappings from data examples, and establish that it performs significantly better than the two baselines.

CCS CONCEPTS

• **Theory of computation → Data exchange**; *Active learning*;

KEYWORDS

GAV schema mappings, active learning, conformance testing

ACM Reference Format:
Balder ten Cate, Phokion G. Kolaitis, Kun Qian, and Wang-Chiew Tan. 2018. Active Learning of GAV Schema Mappings. In *PODS'18: 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3196959.3196974

PODS'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4706-8/18/06…$15.00
https://doi.org/10.1145/3196959.3196974

1 INTRODUCTION

Data exchange is the problem of translating data structured under a source schema into data structured under a target schema, so that the target data is an accurate representation of the source data. Data integration is the problem of combining heterogeneous data from different sources, so that a unified view of the data residing in these sources can be obtained. During the past fifteen years, the modeling and implementation of data exchange and data integration have been facilitated by the systematic use of schema mappings, which are high-level specifications of the relationship between two database schemas [8, 11, 28, 31]. Schema mappings are typically specified via constraints in some suitable logic-based schema-mapping language. The language of GLAV (Global-and-Local-As-View) constraints is the most widely used such formalism. As an important case, it contains the language of GAV (Global-As-View) constraints that populate a relation in the target schema with the result on a conjunctive query over the source schema (see Section 2 for the precise definitions).

Real-world applications can benefit from the use of schema mappings. For example, suppose that company A is being merged with company B and, furthermore, each of these companies has its own legacy database, which means that the schemas of the legacy databases cannot be changed. In such a case, the data residing in the database of company A have to be correctly translated into data that conform to the schema of the database of B, so that a user can obtain information about company A from the database of company B. An appropriately designed schema mapping between the two schemas would facilitate the subsequent data transformation process. Schema mappings are also widely used in such applications as online travel booking websites that provide a unified view of travel-related information (e.g., flights, car rentals, hotels) obtained from multiple data sources with different schemas.

Designing and refining schema mappings manually can be a labor-intensive and error-prone process. To facilitate the design of schema mappings, early work focused on developing graphical interface systems that would derive a schema mapping from correspondences between elements of two schemas marked by a human expert (e.g., [13, 25, 33]). This approach has certain shortcomings, including the fact that multiple non-equivalent schema mappings may be compatible with the same set of correspondences.

More recently, data examples have been used as the basis of alternative approaches to schema-mapping design [1, 16–18, 24]. Here, a data example is a pair (I, J) consisting of a source instance

I and a target instance J. Discovering a schema mapping from data examples is a preferable approach to schema-mapping design because data examples provide a partial description of the underlying semantics between a source schema and target schema.

Several different frameworks for discovering schema mappings from data examples exist, including the fitting framework [1, 2], the repair framework [18, 24], and the learning framework [17]. These three frameworks approach the schema-mapping discovery problem from different angles. Concretely, the fitting framework focuses on the following decision problem: given a finite set E of data examples, determine whether there is a GLAV mapping $\mathcal{M}$ that *fits* E (here, fitting means that the examples in E are *universal* examples for $\mathcal{M}$, i.e., they are "most general" examples for $\mathcal{M}$ - see Section 2 for the precise definition). The repair framework uses a cost model to measure the quality of a schema mapping derived from a single data example; thus, schema-mapping discovery is cast as an optimization problem aiming to produce, given a data example, a schema mapping of minimum cost. The learning framework casts schema-mapping discovery as a computational learning problem aiming to identify a *goal* schema mapping by asking queries (e.g., asking for random data examples of the goal mapping) to various oracles that have access to the goal schema mapping. The algorithms in the fitting framework and the learning framework involve interaction with the user or the learner. More recently, a different framework, called Interactive Mapping Specification (IMS), was developed in [16]. Starting with an initial set E of data examples, IMS derives a GLAV mapping $\mathcal{M}$ through a sequence of interactions with a non-expert user. This derived schema mapping $\mathcal{M}$ has the advantage of often being compact in terms of size and easy to comprehend, but it comes with no quality guarantees; in particular, it may not even be satisfied by the examples in E.

Much is known about the computational complexity of discovering schema mappings in the fitting, the repair, and the learning frameworks. In particular, a fairly complete picture has been developed about the complexity of discovering GAV mappings, i.e., schema mappings specified by GAV constraints. In the fitting framework, the existence of a GAV mapping that fits a set of data examples is a DP-complete problem [1]. In the repair framework and unless NP = RP, there is no polynomial-time algorithm that, given a set of data examples, computes a schema mapping whose cost is bounded by some fixed polynomial factor in the cost of the optimal GAV mapping [18]. In the learning framework, there is a polynomial-time[1] algorithm for learning GAV mappings in Angluin's exact learning model [17].

Concerning tools, there are systems for schema-mapping discovery of GLAV mappings in the fitting and in the IMS frameworks [2, 16]; there is also a system for producing approximately optimal schema mappings in the repair framework, but this system only covers the very restricted class of single-head LAV mappings [18]. So far, there are no tools for schema-mapping discovery in the learning framework.

Our main aim in this paper is to design and evaluate a practical algorithm for learning GAV mappings. As mentioned earlier, ten Cate, Dalmau, and Kolaitis [17] designed a polynomial-time algorithm, called exactGAV, for learning GAV mappings in Angluin's exact learning model [5]. In this model, schema mappings are viewed as concepts; the task is to identify a goal GAV mapping $\mathcal{M}$ by asking *membership* queries (or, more generally, *labeling* queries) and *equivalence* queries. When presented with a source instance I, a labeling query returns the canonical universal solution J for I with respect to $\mathcal{M}$, while an equivalence query tells whether or not some hypothetical GAV mapping $\mathcal{M}'$ is logically equivalent to $\mathcal{M}$. The ability to answer such queries is often described as having a labeling oracle and an equivalence oracle.

How can a practical algorithm for learning a concept be obtained from a theoretical result about exact learning with membership (or labeling) queries and equivalence queries? This issue has been studied since the early days of the exact learning model and, in particular, since Angluin's result [4] that deterministic finite automata can be learned using membership and equivalence queries. The availability of an oracle for membership queries has been interpreted as the existence of a *black box* that implements the behavior of the concept to be learned. Answering equivalence queries, however, is a task of different nature because an equivalence oracle cannot be realized without actual knowledge of the specification of the goal concept. In view of this, substitutes of the equivalence oracle have been introduced and investigated. Angluin [4, 5] showed that if a black box for answering membership queries is available, then the black box can be used to simulate a stochastic equivalence oracle, so that a result about exact learning with membership queries and equivalence queries implies a result about learning in Valiantś [39] model of probably approximate correct (PAC) learning. In a different vein, *conformance testing* has been used as a substitute of the equivalence oracle. In this approach, the equivalence oracle is replaced by a finite set C of membership tests. In particular, conformance testing has been extensively used in learning various classes of automata (see, e.g., [7, 36] and the recent survey by Vaandrager [38]). Note that conformance testing had been used earlier in verification, as well as in black box checking [34].

In this paper, we design and evaluate an algorithm, which we call GAVLearn, for learning GAV mappings. Our algorithm improves on an Occam algorithm that can be extracted from the exactGAV algorithm in [17]. GAVLearn uses a black box that implements the labeling oracle and takes as input a finite set of data examples that constitute the conformance test during the execution of the algorithm. GAVLearn is an active learning algorithm, because, in the words of Vaandrager [38], it accomplishes its task by "actively doing experiments (tests) on the software" (in our case, the software is the black box implementing the membership oracle). The black box could also be interpreted as being a user, in which case GAVLearn can be viewed as an interactive algorithm for discovering GAV mappings. In a different vein, GAVLearn can be construed as solving a *reverse-engineering* problem related to schema-mapping discovery. In reverse engineering, the main task is to discover the specification of a "goal" device from its behavior. In our setting this amounts to the following problem. Suppose that there is a black box that performs data exchange according to some GAV mapping $\mathcal{M}$, the specification of which is unknown due, for example, to proprietary reasons. Is there a way to recover the specification of the schema mapping $\mathcal{M}$ hidden in the black box using the black box and a finite set of data examples?

[1]The notion of a polynomial-time algorithm in the exact learning model requires careful formulation - see Section 2 for the precise definition.

We analyze rigorously the properties of GAVLearn and, among other things, we show that it is an optimal Occam algorithm for learning GAV mappings, which implies that GAVLearn also gives rise to a PAC learning algorithm. In addition to the analysis, we implemented GAVLearn and carried out an extensive experimental evaluation using schema-mapping scenarios produced by iBench [10], a state-of-the-art primitive-based benchmarking tool for information integration tasks. This experimental implementation was done in two parts. We first evaluated GAVLearn on its own and then we compared GAVLearn with two baselines: the GAV-Fit algorithm and the TALOS system. The GAV-Fit algorithm is a special case of the GLAV fitting algorithm introduced and evaluated in [1, 2], while the TALOS system was developed to reverse-engineer conjunctive queries [37]. There is a tight correspondence between unions of conjunctive queries and GAV mappings, which makes it possible to also use the TALOS to discover GAV mappings. Our experimental results show that GAVLearn performs significantly better than these two baselines.

2 PRELIMINARIES AND EARLIER WORK

Schemas and Instances A *schema* $\mathbf{R}$ is a set $\{R_1, \ldots, R_k\}$ of relation symbols. An $\boldsymbol{R}$*-instance* can be identified with the 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 in the relation R_i^I of I that interprets R_i. The *active domain* $\mathrm{adom}(I)$ of an instance I is the set of all values occurring in the facts of I. A *homomorphism* $h : I_1 \to I_2$ is a function from $\mathrm{adom}(I_1)$ to $\mathrm{adom}(I_2)$ such that if $P(a_1, \ldots, a_m)$ is a fact of I_1, then $P(h(a_1), \ldots, h(a_m))$ is a fact of I_2. The *direct product* $I \times I'$ of two $\mathbf{S}$-instances I and I' is the $\mathbf{S}$-instance with facts $S_i(\langle a_1, a_1' \rangle, \ldots, \langle a_k, a_k' \rangle)$, where $S_i(a_1, \ldots, a_k)$ is a fact of I and $S_i(a_1', \ldots, a_k')$ is a fact of $S_i^{I'}$.

Schema Mappings Let $\mathbf{S}$ and $\mathbf{T}$ 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 constraints that are typically expressed in some fragment of first-order logic. In this paper, we focus on constraints that are expressed in the language of *GAV (Global-As-View) constraints*, which is an important special case of the language of *GLAV (Global-and-Local-As-View) constraints*. A GAV constraint is a first-order logic sentence of the form:

$$\forall \mathbf{x}\Big(\varphi(\mathbf{x}) \to T(\mathbf{x})\Big),$$

where $\varphi(\mathbf{x})$ is a conjunction of atoms over the source schema $\mathbf{S}$ and $T(\mathbf{x})$ is a single atom over the target schema $\mathbf{T}$. For succinctness, we will often drop the universal quantifiers when writing constraints. A *GAV mapping* is a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a finite set of GAV constraints.

Example 1. The triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S} = \{E\}$, $\mathbf{T} = \{F\}$, and $\Sigma = \{E(x, y) \wedge E(y, z) \to F(x, z)\}$, is a GAV mapping. □

Solutions, Universal Solutions, and Data Examples. From now on, we assume that the source schema $\mathbf{S}$ and the target schema $\mathbf{T}$ are fixed. A *source instance* is an instance over the source schema $\mathbf{S}$, and a *target instance* is an instance over the target schema $\mathbf{T}$. A *data example* as a pair (I, J) of a source instance I and a target instance J.

A target instance J is a *solution* for a source instance I 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 $\mathcal{M}$. 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 a source instance and a target instance, J is a solution for I w.r.t. $\mathcal{M}$ if and only if 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 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., $h(a) = a$ 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) [1]. Universal solutions are considered to be the preferred solutions for data exchange [20]. Intuitively, a universal solution contains only information that necessarily belongs to every solution of the given source instance. 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 w.r.t. $\mathcal{M}$. We call this target instance the *canonical universal solution of I w.r.t. $\mathcal{M}$* and denote it by $\texttt{can-sol}_{\mathcal{M}}(I)$. For a fixed schema mapping $\mathcal{M}$, the canonical universal solution $\texttt{can-sol}_{\mathcal{M}}(I)$ can be computed in polynomial time in the size of a given source instance I, using the chase procedure [20].

Universal Examples. Let (I, J) be a data example and $\mathcal{M}$ a GAV mapping. We say that (I, J) is a *positive example* for $\mathcal{M}$ if J is a solution of I w.r.t. $\mathcal{M}$; otherwise, (I, J) is a *negative example*. We say that (I, J) is a *universal example* for $\mathcal{M}$ if J is a universal solution for I w.r.t. $\mathcal{M}$. Note that a data example (I, J) that is universal for a GAV mapping $\mathcal{M}$ need not be a canonical universal example for $\mathcal{M}$. Here, we focus on the learnability of GAV mappings using canonical universal examples. There are two reasons for this choice. First, as pointed out in [3], learnability of GAV mappings using universal examples is equivalent to learnability using positive and negative examples; second, for a GAV mapping $\mathcal{M}$, the canonical universal example of a source instance I is the core of the universal solutions for I w.r.t. $\mathcal{M}$. In the rest of the paper, when we say that a data example (I, J) is universal for a GAV mapping $\mathcal{M}$, we mean that J is the canonical universal solution for I w.r.t. $\mathcal{M}$.

Example 2. (Example 1 continued) Given a source instance $I = \{E(a, b), E(b, c)\}$, consider the target instances:

$$J_1 = \{F(a, c)\},\ J_2 = \{F(a, c), F(N_1, N_2)\},$$
$$J_3 = \{F(a, c), F(b, b)\},$$

where N_1, N_2 are nulls (i.e., values that are not in $\mathrm{adom}(I)$). Then (I, J_1) is a canonical universal example for $\mathcal{M}$, whereas (I, J_2) is a universal example for $\mathcal{M}$, but not a canonical universal one. Also, (I, J_3) is a positive example for $\mathcal{M}$, but not a universal one, because J_3 is a solution, but not a universal solution for I w.r.t. $\mathcal{M}$. Finally, if an instance J' excludes the fact $F(a, c)$, then J' is not a solution for I w.r.t. $\mathcal{M}$. □

Computational learning models. We will be interested in the following two major learning models.

(1) The *exact learning model*, introduced by Angluin [5];
(2) The *PAC (probably-approximately-correct) model*, introduced by Valiant [39].

To describe these models, we need some auxiliary notions.

Labeled examples and concepts. 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 cases, $L = \{0, 1\}$ (i.e., positive and negative examples).

A *concept over* X is a function $c : X \to L$, while *a concept class* C is a collection of concepts. We assume that concepts are specified by some representation system, where a *representation system for a concept class* C is a string language $\mathcal{L}$ over some finite alphabet, together with a surjective function $r : \mathcal{L} \to C$. For every concept representation $l \in \mathcal{L}$, we write $size(l)$ to denote its length, by which we mean the number of bits needed to write l using a suitable binary encoding. For example, if $X = \{0,1\}^n$ is the set of all n-ary boolean assignments, then we can consider the concept class C consisting of all Boolean functions $c : \{0,1\}^n \to \{0,1\}$ specified by Boolean formulas expressed as DNF (disjunctive normal form) formulas with 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 returns its label $c(x)$ in L, according to c. Note that when $L = \{0,1\}$, labeling oracles are also known as *membership* oracles.

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 $c \equiv c'$ (i.e., c and c' represent the same concept); otherwise, it returns a counterexample x (i.e., an example x such that $c(x) \neq c'(x)$).

Exact learning. Following Angluin [5], an algorithm `alg` *exactly learns a concept class* C *with labeling and/or equivalence queries* if, for all natural numbers n and for all concepts $c \in C$ 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. Later, Angluin [6] introduced the following notion of efficient exact learnability. A concept class 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 C that runs in polynomial time in the following sense: there is a bivariate polynomial $p(n,m)$ such that at any point during a run of the algorithm, the time used by the algorithm up to that point (counting one step per oracle call) is bounded by $p(n,m)$, where n is the size of the goal concept and m is the size of the largest counterexample returned by calls to the equivalence oracle up to that point ($m = 0$ if no equivalence queries have been used). This is a delicate notion of efficiency that avoids the pitfall of allowing for an "efficient" algorithm that first finds the goal concept via exhaustive search and then forces the oracle to produce very large counterexamples to make up at the end for the time spent on the exhaustive search.

PAC learning. The task of a PAC learning algorithm is to approximately learn an unknown goal concept $c \in C$ 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 *probability distribution* over X is a function $D : X \to [0,1]$ such that $\Sigma_{x \in X} D(x) = 1$. For every concept $c \in C$ and probability distribution $D : X \to [0,1]$, we denote by $\text{EX}_{c,D}$ 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.

An algorithm `alg` *PAC learns a concept class* C if for all rational numbers 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 C, when `alg` is run with input ϵ, δ, n and example oracle $\text{EX}_{c,D}$, it outputs with probability at least $1-\delta$ a concept h with error $\text{ERR}_{c,D}(h) \leq \epsilon$. Here, $\text{ERR}_{c,D}(h) = \mathbf{Pr}_{x \in D}(c(x) \neq h(x))$, where $\mathbf{Pr}_{x \in D}(c(x) \neq h(x))$ denotes the probability of the event $c(x) \neq h(x)$ with $x \in X$ drawn from D.

A concept class C is *PAC learnable* if there is a PAC algorithm `alg` that learns it. If a concept class can be learned by a PAC algorithm with access to labeling oracle, then the concept class C is said to be *PAC learnable with labeling queries*.

Occam learning and its relation to PAC learning. Occam learning is an important model of algorithmic learning [12].

Definition 3. *Let C be a concept class. An* Occam algorithm for C with parameters $0 \leq \alpha < 1$ and $k \geq 1$ *is an algorithm A that takes as input a collection*

$$(x_1, c(x_1)), \ldots, (x_m, c(x_m))$$

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 $n = |c|$ is the length of c. If $\alpha = 0$ and $k = 1$, then A is an optimal Occam algorithm *for C.*

Theorem 4. *[12] If C is a concept class for which an Occam algorithm exists, then C is PAC learnable.*

In particular, if A is an Occam algorithm for C, then we can obtain a PAC learning algorithm A' for C by asking for m many random examples, inputting these random examples to A, and outputting the hypothesis returned by A, where

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

2.1 Earlier Work on Learning GAV Mappings

Fix a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a GAV mapping, where Σ is a finite set of GAV constraints over $\mathbf{S}$ and $\mathbf{T}$. Semantically, $\mathcal{M}$ can be identified with the set of all its positive examples, i.e., the set of all pairs (I, J), where I is a source instance and J is a solution for I w.r.t. $\mathcal{M}$. It is also easy to see that $\mathcal{M}$ can also be identified with the set of all its universal examples, i.e., the set of all pairs $(I, \texttt{can-sol}_{\mathcal{M}}(I))$, where $\texttt{can-sol}_{\mathcal{M}}(I)$ is the canonical solution for I w.r.t. $\mathcal{M}$ (if two GAV mappings have the same universal examples, then they are logically equivalent). Thus, $\mathcal{M}$ can be viewed as a concept given by the function c that maps every source instance I to the label $\texttt{can-sol}_{\mathcal{M}}(I)$. We shall denote by GAV(**S**,**T**) the concept thus defined.

In [17], ten Cate, Dalmau, and Kolaitis studied GAV mappings in the exact learning model and the PAC model. The following results from [17] will be of interest to us here.

Theorem 5. *[17] Let **S** be a source schema and **T** a target schema. If **S** contains a relation symbol of arity at least two, then the concept class GAV(**S**,**T**) is not efficiently exactly learnable with labeling queries.*

Theorem 5 implies that there is no efficient exact learning algorithm for our motivating reverse engineering problem. However, it

was also shown in [17] that if both a labeling oracle and an equivalence oracle are available, then GAV(**S**,**T**) is efficiently exactly learnable.

Theorem 6. *[17] Let **S** be a source schema and **T** a target schema. The concept class GAV(**S**,**T**) is efficiently exactly learnable with labeling and equivalence queries.*

Theorem 6 and Angluin's definition of exact learnability imply that, for every goal GAV mapping $\mathcal{G}$, both the number of labeling queries and the number of equivalence queries needed to learn $\mathcal{G}$ is bounded by a polynomial in the size of $\mathcal{G}$ and the size of the largest counterexample used in the exact learning algorithm in Theorem 6. It is worth comparing and contrasting Theorem 6 with a related result obtained by Khardon in [26] for *universal Horn theories*, that is, finite sets of universally quantified Horn formulas over a relational schema. A GAV mapping $\mathcal{G} = (\mathbf{S}, \mathbf{T}, \Sigma)$ can be viewed as the special case of a universal Horn theory over the schema $\mathbf{S} \cup \mathbf{T}$ in which the two schemas **S** and **T** have no relation symbols in common, the relation symbols from **S** occur only in the antecedent of each Horn formula, and the relation symbols from **T** occur only in the conclusion of each Horn formula. In [26], it was shown that universal Horn theories are exactly learnable with equivalence and membership queries, where a *membership query* asks whether a universal Horn theory is satisfied in a given instance. As pointed out in [17], for GAV schema mappings, membership queries can be simulated using labeling queries: (I, J) satisfies $\mathcal{G}$ if and only if the universal solution of I with respect to $\mathcal{G}$ is contained in J. Hence, Khardon's result implies that the concept class GAV(**S**,**T**) is exactly learnable with labeling and equivalence queries. However, in contrast with Theorem 6, the learning algorithm for universal Horn theories given in [26] is not an efficient learning algorithm. Actually, in Khardon's algorithm, both the number of membership queries and the number of equivalence queries are exponential in the number of variables of the goal universal Horn theory (and hence exponential in the size of that theory). Exact learnability has also been studied in the context of description logic ontologies: in [29], the authors study exact learnability with membership and equivalence queries, for the description logics DL-Lite and EL; while [30] studies exact learnability with tuple-membership queries and logical entailment queries, for the same description logics. DL-Lite and EL, viewed as constraint languages, are incomparable (that is neither subsuming, nor subsumed by) the language of GAV constraints.

In [17], a concrete algorithm, which we call exactGAV algorithm, was presented that efficiently and exactly learns GAV schema mappings using labeling queries and equivalence queries. The exactGAV algorithm, however, is not applicable to our motivating problem because, although the labeling oracle can be implemented by the black box provided, the equivalence oracle required by exactGAV is missing and it is impossible to implement it using the available resources. Concretely, to tell whether a candidate mapping $\mathcal{H}$ and a goal mapping $\mathcal{G}$ (given as a black box) are logically equivalent, one has to check if the two mappings produce the same target instance for every input source instance. Angluin showed in [5] that a learning method that uses equivalence queries and achieves exact learnability may be modified to achieve PAC learnability. In [17], the following corollary showing the existence of such a PAC learning algorithm for GAV(**S**,**T**) is given.

Corollary 7. *[17] Let **S** be a source schema and **T** a target schema. The concept class GAV(**S**,**T**) is PAC learnable with labeling queries using a polynomial number of calls to the example oracle and the labeling oracle.*

In the next section, we present a concrete algorithm, which we call GAVLearn, for the concept class GAV(**S**,**T**). The GAVLearn algorithm improves on an Occam algorithm that can be extracted from the exactGAV algorithm in [17]. We show that GAVLearn is an optimal Occam algorithm for GAV(**S**,**T**); hence, by Theorem 4, it gives rise to a PAC learning algorithm for GAV(**S**,**T**).

3 THE GAVLEARN ALGORITHM

This section contains the description and analysis of the GAVLearn algorithm. We first introduce several concepts and state several lemmas, then we present the GAVLearn algorithm and show that it is an Occam algorithm for the concept class GAV(**S**,**T**). Due to space limitations, most proofs are omitted or only sketched. Details are in the Appendix.

Description and Properties of GAVLearn The basic idea behind GAVLearn is to iteratively maintain a GAV mapping $\mathcal{H}$ consisting of constraints that are *critically sound* with respect to the goal mapping $\mathcal{G}$; the algorithm terminates when $\mathcal{H}$ fits the given set E of data examples. A GAV constraint C is *sound* with respect to $\mathcal{G}$ if $\mathcal{G}$ logically implies C, that is, for every example (I, J) that satisfies $\mathcal{G}$, we have that (I, J) also satisfies C. A GAV constraint C is *critically sound* with respect to $\mathcal{G}$ if (i) C is sound with respect to $\mathcal{G}$; and (ii) 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'.

The following definition is important in the sequel.

Definition 8. *Let $C : \phi \rightarrow \psi$ be a GAV constraint. The* canonical source instance *of C is the instance I_ϕ with the variables of ϕ as elements and the conjuncts of ϕ as facts. The* canonical target instance *of C is the instance J_ψ obtained from ψ in the same way (and consisting of a single fact). The GAV constraint C is called the* canonical GAV constraint *of the two instances I_ϕ, J_ψ.*

As in [17], a GAV constraint $\phi \rightarrow \psi$ will often be identified with the pair (I_ϕ, J_ψ). A *homomorphism* $\widehat{h} : C \mapsto C'$ between two GAV constraints C and C' is a function that homomorphically maps atomic formulas occurring in the left-hand side (respectively, in the right-hand side) of C to atomic formulas occurring in the left-hand side (respectively, in the right-hand side) of C'. We write $C \mapsto C'$ to denote the existence of a homomorphism from C to C'. The next lemma in [17] generalizes results by Chandra and Merlin [19].

Lemma 9. *[17] For all GAV constraints $C = (I, \{f\})$, the following statements are equivalent: (1) $\mathcal{G}$ logically implies C; (2) $C' \mapsto C$ for some $C' \in \mathcal{G}$; (3) $f \in \texttt{can-sol}_{\mathcal{G}}(I)$.*

The next lemma, which is a direct generalization of a lemma in [17], lies at the core of the GAVLearn algorithm.

Lemma 10. *Let $\mathcal{G}$ be a GAV mapping. Recall that the labeling oracle $LABEL_{\mathcal{G}}$ takes as input a source instance I and returns the target instance can-sol$_{\mathcal{G}}(I)$. If $C = (I, \{f\})$ is a GAV constraint that is logically implied by $\mathcal{G}$, then, with access to $LABEL_{\mathcal{G}}$, we can construct in polynomial time a GAV constraint $Crit_{\mathcal{G}}(C) = (I', \{f\})$ with $I' \subseteq I$ such that $Crit_{\mathcal{G}}(C)$ is critically sound with respect to $\mathcal{G}$.*

Proof. To compute $Crit_{\mathcal{G}}(C)$, we 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 sub-instance obtained from I. We stop when a minimal sub-instance I' of I is reached. By construction, the constraint $(I', \{f\})$ is critically sound with respect to $\mathcal{G}$. Note that there are $|I|$ many facts, hence at most $|I|$ many iterations are needed. □

The next lemma uses a direct-product construction for GAV constraints. Let $C = (I, \{f\})$ and $C' = (I', \{f'\})$ be two GAV constraints, where the target facts f and f' use the same target relation. We write $C \times C'$ to denote the GAV constraint $C'' = (I'', \{f''\})$, where I'' is the direct product of I and I', and where f'' is the direct product of f and f', provided that such a GAV constraint C'' exists. For example, if

$$C_1 : R(x, x, y) \rightarrow T(x) \quad \text{and} \quad C_2 : R(x, y, y) \rightarrow T(x),$$

then the direct product $C_1 \times C_2$ is

$$E(z_{\langle x,x\rangle}, z_{\langle x,y\rangle}, z_{\langle y,y\rangle}) \rightarrow T(z_{\langle x,x\rangle}),$$

where (as the notation $z_{\langle u,v\rangle}$ used above is meant to suggest) each variable in the direct product $C \times C'$ corresponds to a pair of variables from C_1 and C_2, respectively; moreover, the facts in $C_1 \times C_2$ are the facts $R(z_{\langle u_1,v_1\rangle}, \ldots, z_{\langle u_n,v_n\rangle})$ such that $R(u_1, \ldots, u_n)$ occurs in C_1 and $R(v_1, \ldots, v_n)$ occurs in C_2 (see [17] for a detailed definition). In general, the direct product $C \times C'$ may not be well defined; indeed, for the GAV constraints $C : E(x, y) \rightarrow T(x)$ and $C' : E(u, v) \rightarrow T(v)$, the direct product $E(z_{\langle x,u\rangle}, z_{\langle y,v\rangle}) \rightarrow T(z_{\langle x,v\rangle})$ is not a well-formed GAV constraint as it violates the condition that each variable occurring in the right-hand side occur also in the left-hand side.

Lemma 11. *[17] Let C, C_1, C_2 be GAV constraints with homomorphisms $C \mapsto C_1$ and $C \mapsto C_2$. Then $C_1 \times C_2$ is well defined, and moreover, $C \mapsto C_1 \times C_2$; $C_1 \times C_2 \mapsto C_1$; and $C_1 \times C_2 \mapsto C_2$.*

The GAVLearn algorithm is depicted in Figure 1. The input consists of a goal GAV mapping $\mathcal{G}$ whose specification is unknown (i.e., it is given as a labeling oracle) and a set E of existing universal examples for $\mathcal{G}$. The GAVLearn algorithm is an iterative learning algorithm, that, in each iteration, checks if the current candidate mapping $\mathcal{H}$ is a fitting GAV mapping for E. If it is, the algorithm terminates and returns $\mathcal{H}$; otherwise, the algorithm finds an example $(I, J) \in E$ that is not universal for $\mathcal{H}$ (Lines 6-8 of GAVLearn). In fact, we will show that it must be the case that $\texttt{can-sol}_{\mathcal{H}}(I) \subsetneq J$, which provides us with a candidate constraint $(I, \{f\})$ with $f \in J \setminus \texttt{can-sol}_{\mathcal{H}}(I)$, that, intuitively, is captured by the goal mapping, but not by the mapping $\mathcal{H}$ learned so far. A new GAV constraint derived from $(I, \{f\})$ is added to $\mathcal{H}$. The step of computing the direct product C' at Line 11 of GAVLearn is important as it serves to compute, when possible, a more general GAV constraint that constitutes a common generalization of a previously

Input: $\mathcal{G}$ - goal mapping (as a labeling oracle);
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** each $(I, J) \in E$ is canonical universal for $\mathcal{H}$ **then**
4: **return** $\mathcal{H}$
5: **end if**
6: choose an $(I, J) \in E$ such that $J \neq \texttt{can-sol}_{\mathcal{H}}(I)$
7: // In the proof of Thm 13, we show $\texttt{can-sol}_{\mathcal{H}}(I) \subsetneq J$
8: $f \leftarrow$ choose a fact $f \in J \setminus \texttt{can-sol}_{\mathcal{H}}(I)$
9: **if** $\mathcal{G}$ logically implies $(I, \{f\}) \times C$ for some $C \in \mathcal{H}$ **then**
10: Choose $C \in \mathcal{H}$ such that $\mathcal{G}$ logically implies $(I, \{f\}) \times C$
11: $\mathcal{H} \leftarrow (\mathcal{H} \setminus \{C\}) \cup \{Crit_{\mathcal{G}}((I, \{f\}) \times C)\}$
12: **else**
13: $\mathcal{H} \leftarrow \mathcal{H} \cup \{Crit_{\mathcal{G}}((I, \{f\}))\}$
14: **end if**
15: **end while**
16: **return** $\mathcal{H}$

Figure 1: Algorithm GAVLearn

learned constraint C and the newly discovered candidate constraint $(I, \{f\})$.

Lines 11 and 14 of GAVLearn include another key step, namely, computing a critically sound constraint $Crit_{\mathcal{G}}(C')$ for a given candidate constraint C' with respect to the goal mapping $\mathcal{G}$. The main idea of this step is to remove relational atoms from the candidate constraint and, thus, to obtain a simpler constraint that is less specific to the input data example. As part of this step, GAVLearn generates additional universal examples. Specifically, when computing a critically sound sub-constraint $Crit_{\mathcal{G}}(C')$ from C' using Lemma 10, the algorithm repeatedly queries the labeling oracle (i.e., the goal mapping $\mathcal{G}$), generating up to k many new universal examples for $\mathcal{G}$, where k is the number of atoms in the left-hand side of C'. As we will see in Section 4.2, this is the key feature that differentiates GAVLearn from GAV-Fit (one of the baselines), because GAV-Fit does not try to simplify the constraints derived from the input set of examples.

Example 12. We give an example run of GAVLearn. Consider a goal mapping $\mathcal{G}$ consisting of the GAV constraints

$$\begin{aligned} S(x, y) \wedge R(y, z) &\rightarrow T(x, z)\ , \\ M(x, y) \wedge N(y, z) &\rightarrow Q(x, y, z)\ . \end{aligned}$$

Let $E = \{(I, J)\}$, where

$$\begin{aligned} I &= \{S(a, b), R(b, c), M(a, b), N(b, c)\}\ , \\ J &= \{T(a, c), Q(a, b, c)\}. \end{aligned}$$

Initially, $\mathcal{H} = \emptyset$. In the first iteration, GAVLearn checks if $\mathcal{H}$ and $\mathcal{G}$ agree on E, that is, if $J = \texttt{can-sol}_{\mathcal{H}}(I)$. In fact, $\texttt{can-sol}_{\mathcal{H}}(I) = \emptyset$ since $\mathcal{H}$ is empty and $\texttt{can-sol}_{\mathcal{H}}(I) \subsetneq J$. GAVLearn selects a fact $f \in J \setminus \texttt{can-sol}_{\mathcal{H}}(I)$. Let us assume that $f = T(a, c)$.

Next, GAVLearn uses Lemma 10 to compute a critically sound sub-constraint $(I', \{f\})$ of the GAV constraint $(I, \{f\})$. In this example, $I' = \{S(a, b), R(b, c)\}$. Indeed, note that, if either $S(a, b)$ or $R(b, c)$ is removed from I, the resulting constraint is not sound, i.e., is not logically implied by $\mathcal{G}$, while $M(a, b)$ and $N(b, c)$ can be removed without affecting soundness. The resulting GAV constraint

$$C_1 \ : \ S(x, y) \wedge R(y, z) \rightarrow T(x, z)$$

is added to the mapping $\mathcal{H}$.

In the second iteration, $\mathcal{H}$ consists of the GAV constraint C_1. The two mappings $\mathcal{H}$ and $\mathcal{G}$ still do not agree on (I, J), as $\texttt{can-sol}_{\mathcal{H}}(I) = \{T(a, c)\}$ and therefore $\texttt{can-sol}_{\mathcal{H}}(I) \subsetneq J$. This time, the fact f that is selected is $Q(a, b, c)$. By the similar procedure described above, a new constraint is obtained, namely

$$C_2 \ : \ M(x, y) \wedge N(y, z) \rightarrow Q(x, y, z) \ ,$$

and the new constraint is added to $\mathcal{H}$.

In the third iteration, $\mathcal{H}$ and $\mathcal{G}$ agree on E, so the algorithm terminates and returns $\mathcal{H}$. Actually, in this example, the GAV mapping $\mathcal{H}$ returned by GAVLearn is identical to $\mathcal{G}$.

Theorem 13. *The algorithm GAVLearn is an optimal Occam algorithm for GAV(**S**,**T**). Specifically, given a labeling oracle for a GAV mapping $\mathcal{G}$, as well as a set E of universal examples for $\mathcal{G}$, the algorithm GAVLearn returns a GAV mapping $\mathcal{H}$ such that E is universal for $\mathcal{H}$ and the size of $\mathcal{H}$ is at most the size of $\mathcal{G}$.*

Proof. *(Outline)* Let $\mathcal{H}_i$ denote 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. The proof of Theorem 13 makes use of the following claims, the proofs of which can be found in Appendix 7.

Claim 14. *For every $i \geq 0$ such that $\mathcal{H}_i$ is defined, $\mathcal{H}_i$ consists of GAV constraints that are sound with respect to $\mathcal{G}$. In particular, in Line 6 of GAVLearn, it must be the case that $\texttt{can-sol}_{\mathcal{H}_i}(I) \subsetneq J$.*

Claim 15. *Let $i \geq 0$ such that $\mathcal{H}_i$ is defined. For every $C \in \mathcal{H}_i$, there is a $C^* \in \mathcal{G}$ such that $C^* \mapsto C$. Furthermore, for each $C^* \in \mathcal{G}$, the set $\mathcal{H}_i(C^*) = \{C \in \mathcal{H}_i \mid C^* \mapsto C\}$ is either empty or is a singleton.*

From Claim 15 we obtain:

Claim 16. *The size of the schema mapping $\mathcal{H}$ produced by GAVLearn is at most the size of $\mathcal{G}$.*

From the description of GAVLearn, it follows that, upon termination, the returned GAV mapping $\mathcal{H}$ fits the universal examples E (i.e., E is universal for $\mathcal{H}$). Next, we show that the algorithm terminates after at most n many iterations, where n is the number of variables occurring in the constraints of $\mathcal{G}$. For every $i \geq 1$ such that $\mathcal{H}_i$ is defined, let $s_i = \Sigma_{T \in \mathcal{G}} s_i^T$, where $s_i^T = 0$ if $\mathcal{H}_i(T)$ is empty, and s_i^T is the number of variables occurring in the unique element of $\mathcal{H}_i(T)$, otherwise (this is well defined by Claim 15). For a GAV constraint C, let $nvar(C)$ be the number of variables occurring in C.

Claim 17. *For every $i > 0$ such that $\mathcal{H}_i$ is defined, we have $s_i > s_{i-1}$, and $s_i \leq n$, where $n = \Sigma_{C \in \mathcal{G}} nvar(C)$.*

Claim 17 implies that GAVLearn will terminate after n iterations of the *while* loop. Together with the fact that the mapping returned $\mathcal{H}$ by GAVLearn fits the universal examples E and the fact that the size of $\mathcal{H}$ is at most n, this concludes the proof of Theorem 13. □

In [23], Gottlob, Pichler, and Savenkov introduced and studied a normal form for GLAV schema mappings, which we call the *GPS normal form*. We omit here the precise definition, but we point out that the GPS normal form has the following main features: it satisfies certain desirable minimality criteria, is unique up to renaming variables, and is used to define semantics of queries in data exchange, so that the semantics does not depend on the syntactic representation of the underlying schema mapping. We also note that schema mappings produced in the aforementioned Interactive Mapping Specification framework [16] are in GPS normal form. From the proof of Theorem 13 (and, more specifically, from Claim 15 and the use of critically sound constraints in the learning algorithm), it follows that the GAV mapping returned by GAVLearn is guaranteed to be in GPS normal form (we do not give the details for lack of space).

Additional Properties of GAVLearn Theorem 13 shows that the GAV mapping produced by GAVLearn fits the input set of data examples, and, moreover, the size of the learned mapping is bounded by the size of the goal mapping. However, it does not yet establish any further relationship between the output GAV mapping and the goal mapping (beyond the aforementioned relationship in size and the fact that both GAV mappings fit the examples in E). In particular, Theorem 13 does not show that the GAV mapping produced by our algorithm is always logically equivalent to the hidden goal mapping, thus it may not generalize well to examples outside of the set E. Establishing a tighter relationship between the learned GAV mapping and the goal mapping requires further assumptions on the choice of the set of universal examples E. We now present two results in this direction.

The first result asserts that given a sufficiently rich set of universal examples, the GAV mapping $\mathcal{H}$ returned by GAVLearn is logically equivalent to the goal mapping $\mathcal{G}$.

Theorem 18. *Let $\mathcal{G}$ be an arbitrary GAV mapping.*

1. *For every set E of universal examples w.r.t. $\mathcal{G}$, on input E and $\mathcal{G}$, GAVLearn returns a GAV mapping that is logically implied by $\mathcal{G}$.*
2. *There exists a finite set of universal examples $E_{\mathcal{G}}$ (w.r.t. $\mathcal{G}$) such that for every set E of universal examples with $E_{\mathcal{G}} \subseteq E$, we have that, on input E and $\mathcal{G}$, GAVLearn returns a GAV mapping that is logically equivalent to $\mathcal{G}$.*

Proof. (*Sketch*) The first item follows from Claim 14. For the second item, let $E_{\mathcal{G}}$ be the *canonical set of universal data examples* for $\mathcal{G}$ [1]. This set contains, for every GAV constraint $\phi \rightarrow \psi$ of $\mathcal{G}$, a data example $(I_\phi, \texttt{can-sol}_{\mathcal{G}}(I_\phi))$. Let $\mathcal{H}$ be the GAV mapping produced by GAVLearn on input E and $\mathcal{G}$, where $E_{\mathcal{G}} \subseteq E$. It follows from the construction of $E_{\mathcal{G}}$, and the fact that $\mathcal{H}$ fits $E_{\mathcal{G}}$, that $\mathcal{H}$ logically implies $\mathcal{G}$. We already showed that $\mathcal{G}$ logically implies $\mathcal{H}$. Therefore, the two GAV mappings are logically equivalent. □

Theorem 18 asserts that, in principle, there is a conformance test such that, if given as input to GAVLearn, the output will be a GAV mapping that is logically equivalent to the hidden goal mapping. However, Theorem 18 does not help us identify such a conformance test because $E_{\mathcal{G}}$ depends on the *unknown* goal GAV mapping $\mathcal{G}$. Nonetheless, if the size or an upper bound on the size of $\mathcal{G}$ is known, then it is possible to construct such a conformance test for learning GAV mappings. This is analogous to the state of affairs in learning finite automata and in black box checking [34].

Note also that the set of examples $E_{\mathcal{G}}$ does not necessarily *uniquely characterize* the GAV mapping $\mathcal{G}$ in the sense of [3], that is, there may exist GAV mappings that fit $E_{\mathcal{G}}$ but are not logically

equivalent to $\mathcal{G}$. This is so because, as shown in [3], not every GAV mapping is uniquely characterizable by a finite set of examples.

The second result assumes that the set E of universal examples consists of sufficiently many random examples. Since Theorem 13 shows that GAVLearn is an Occam algorithm with $\alpha = 0$ and $k = 1$ (recall Definition 3), we have that Theorems 4 and 13 together imply the following corollary [2].

Corollary 19. *Assume that $\mathcal{G}$ is a goal GAV mapping 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. If*

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

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

Here, $\mathsf{ERR}_{\mathcal{G},D}(\mathcal{H})$ denotes the expected error rate of the mapping $\mathcal{H}$ with respect to $\mathcal{G}$ and D. Concretely,

$$\mathsf{ERR}_{\mathcal{G},D}(\mathcal{H}) = \mathop{\mathbf{Pr}}_{x \in D}(\mathcal{G}(x) \neq \mathcal{H}(x)),$$

i.e., it is the probability of the event $\mathcal{G}(x) \neq \mathcal{H}(x)$ when x is a source instance drawn from the distribution D. Recall also that $\mathcal{G}(x)$ (resp., $\mathcal{H}(x)$) denotes $\mathsf{can\text{-}sol}_{\mathcal{G}}(x)$ (resp., $\mathsf{can\text{-}sol}_{\mathcal{H}}(x)$).

Corollary 19 implies that GAVLearn is a PAC learning algorithm. Note that GAVLearn does not run in polynomial time because the canonical universal solution for a source instance I with respect to a candidate schema mapping $\mathcal{H}$ is not computable in polynomial time, in general (because the schema mapping $\mathcal{H}$ is not fixed in GAVLearn). However, we can show that the number of calls to the labeling oracle is polynomial (see the Appendix for the proof):

Theorem 20. *GAVLearn asks at most*

$$size(\mathcal{G})^2 \cdot maxsize(E)$$

many labeling queries, where $size(\mathcal{G})$ is the size of the goal schema mapping, and $maxsize(E)$ is the maximum number of facts in an input data example.

The *expected error rate* $\mathsf{ERR}_{\mathcal{G},D}(\mathcal{H})$ used in the statement of Corollary 19 is a rather rough metric for measuring the performance of the GAV mapping $\mathcal{H}$ with respect to the goal mapping $\mathcal{G}$ and the distribution D. More refined metrics are the *precision*, the *recall*, and the *F-score*. Given a source instance I, the *precision* of $\mathcal{H}$ on I (with respect to goal mapping $\mathcal{G}$) is the fraction of tuples in $\mathsf{can\text{-}sol}_{\mathcal{H}}(I)$ that are contained in $\mathsf{can\text{-}sol}_{\mathcal{G}}(I)$, while the *recall* of $\mathcal{H}$ is the fraction of tuples of $\mathsf{can\text{-}sol}_{\mathcal{G}}(I)$ that are included in $\mathsf{can\text{-}sol}_{\mathcal{H}}(I)$. By convention, if $\mathsf{can\text{-}sol}_{\mathcal{H}}(I) = \emptyset$, the precision of $\mathcal{H}$ on I is said to be 1.0, while, if $\mathsf{can\text{-}sol}_{\mathcal{G}}(I) = \emptyset$, the recall of $\mathcal{H}$ on I is said to be 1.0. The *F-score* of $\mathcal{H}$ on a source instance I (with respect to $\mathcal{G}$) is the harmonic mean of precision and recall, i.e., it is the quantity

$$(2 \cdot precision \cdot recall) \ / \ (precision + recall) \ .$$

Let $\mathcal{H}$ be a GAV mapping produced by GAVLearn. It is easy to see that the expected recall of $\mathcal{H}$ on instances drawn from D is at least $1 - \mathsf{ERR}_{\mathcal{G},D}(\mathcal{H})$. From Theorem 18(1), it follows that the expected precision of $\mathcal{H}$ on D is 1. Therefore, the expected F-score is at least $\frac{2(1-\mathsf{ERR}_{\mathcal{G},D}(\mathcal{H}))}{2-\mathsf{ERR}_{\mathcal{G},D}(\mathcal{H})}$, which, in turn, is at least $1-\mathsf{ERR}_{\mathcal{G},D}(\mathcal{H})$ since $0 \leq \mathsf{ERR}_{\mathcal{G},D}(\mathcal{H}) \leq 1$. Thus, the expected F-score of $\mathcal{H}$ on instances drawn from D is at least $1-\mathsf{ERR}_{\mathcal{G},D}(\mathcal{H})$, which means that Corollary 19 provides also a lower bound on the expected F-score of $\mathcal{H}$. [3]

Comparison with Other Algorithms As discussed in Section 2, an efficient exact learning algorithm for GAV(**S**,**T**), using labeling and equivalence queries, was presented in [17]. As a corollary, it was shown that GAV(**S**,**T**) is PAC learnable with labeling queries. GAVLearn improves in multiple ways upon the PAC learning algorithm that can be extracted from the proof of this corollary: it is guaranteed to produce a GAV mapping whose size is at most the size of the goal mapping; it is guaranteed to produce a GAV schema mapping in GPS normal form; and the bound on the number of examples obtained in Corollary 19 is linear in n. These improvements reflect a subtle difference between GAVLearn and the learning algorithm in [17]: in Line 11 of GAVLearn, a constraint C is removed from the set $\mathcal{H}$. In contrast, the learning algorithm in [17] is monotonic, in the sense that constraints are only added to the hypothesis, never removed. It is for this reason that the learning algorithm in [17] may generate schema mappings whose size is arbitrarily large compared to the size of the goal schema mapping.

Recall also from Section 2 that an exact learning algorithm for universal Horn theories, using membership and equivalence queries, was given in [26]. In [9], that learning algorithm was adapted into an algorithm, called LOGAN-H, for deriving a universal Horn theory from an input set of positive and negative examples without the use of any oracle. Similarly to our approach, equivalence queries were replaced by the use of the input set of examples as a conformance test. Moreover and quite interestingly, membership queries were also replaced by an approximate membership test based solely on the input set of examples. As discussed in Section 2, GAV mappings are a special case of universal Horn theories. One may therefore ask whether LOGAN-H could be used for deriving a GAV schema mapping from a set of positive and negative examples. Unfortunately, it turns out that, when used for this purpose, LOGAN-H may produce a GAV mapping that is not logically implied by the goal mapping (even though the input examples are consistent with the goal schema mapping). Indeed, let

$$\mathcal{G} = (\{R\}, \{T_1, T_2\}, \{R(x) \rightarrow T_1(x)\}) \ .$$

Given as input a positive example $(\{R(a)\}, \{T_1(a), T_2(a)\})$ and a negative example $(\{R(a)\}, \emptyset)$, the output of LOGAN-H will include both $R(x) \rightarrow T_1(x)$ and $R(x) \rightarrow T_2(x)$. Furthermore, it can be shown that the output of LOGAN-H may grow arbitrarily with the size of the input examples and, in general, is not bounded by any function in the size of the goal mapping alone.

4 EXPERIMENTAL EVALUATION

A typical methodology for evaluating a learning algorithm for schema-mapping discovery involves the following steps. Given

[2] This corollary can also be derived from Angluin's result in [5] to the effect that learnability in her exact model with labeling and equivalence queries implies PAC learnability with labeling queries.

[3] One might wonder if the definition of expected F-score used here is reasonable, at it works in our favor for instances I with $\mathsf{can\text{-}sol}_{\mathcal{G}}(I) = \emptyset$. An alternative definition of expected F-score would only pertain to instances I for which $\mathsf{can\text{-}sol}_{\mathcal{G}}(I) \neq \emptyset$. In this case, Corollary 19 can still be applied, not directly to the distribution D, but to the distribution D' that is the restriction of D to such instances I.

a goal schema mapping $\mathcal{G}$ and a set S of universal examples for $\mathcal{G}$, the first step is to split S into a training set E and a test set V. The next step is to run the learning algorithm on E and derive a schema mapping $\mathcal{H}$ that is then evaluated on V. The quality of $\mathcal{H}$ with respect to V and $\mathcal{G}$ is evaluated using precision, recall, and F-score that are defined in the previous section. We extend the definitions of these metrics to a set of data examples as follows: the precision (respectively, recall) of $\mathcal{H}$ on a set V of test examples is the average precision (respectively, recall) of $\mathcal{H}$ over all examples in V. The F-score of $\mathcal{H}$ on V is computed with respect to the average precision and average recall on V.

In the sequel, we present an experimental evaluation of GAVLearn, as well as a comparison with two baselines: (i) the GAV fitting algorithm, denoted by GAV-Fit, which is a special case of the GLAV fitting algorithm introduced in [1]; (ii) the TALOS system introduced in [37]. In view of Theorem 18 (1), we focus on the recall in the stand-alone evaluation of GAVLearn, but we also consider F-scores when comparing GAVLearn to the two baselines.

At present, there are no established benchmarks for evaluating and comparing tools for schema-mapping discovery. Our experimental evaluation of GAVLearn is carried out using iBench, which is a metadata generator capable of creating schema mappings of varying complexity [10]. We note that iBench was also used to evaluate the Interactive Mapping Specification (IMS) tool developed in [16] and mentioned in the Introduction of the present paper. We use iBench to create both goal schema mappings and universal data examples. iBench is a primitive-based metadata generator that can create schema mappings of different characteristics using various parameterized primitives. In particular, the following primitives of iBench are relevant for GAV constraints.

- `COPY` primitive (copying): e.g., $S(x) \rightarrow T(x)$.
- `DelAttr` primitive (projection): e.g., $S(x, y) \rightarrow T(x)$.
- `Merge` primitive (join multiple source atoms to create one target atom): e.g., $S_1(x, y) \wedge S_2(y, z) \rightarrow T(x, y, z)$.

The following parameters can further refine these primitives:

- `JoinSize`: number of joining source atoms per merge primitive;
- `SourceShare`/`TargetShare`: specifies the percentage of primitives that share the same source/target relation;
- `JoinKind`: type of joins (e.g., star or chain) for merge primitives.

There are other parameters (e.g., arity of a source relation) that can be tuned to create different GAV constraints. We kept the default values as these parameters are less important than the ones above.

Schema-mapping generation. We used iBench to create three different types of GAV mappings: *simple, moderate*, and *complex*. Concretely, a simple GAV mapping contains ten GAV constraints in total: three `COPY`, three `DelAttr`, and four `Merge` constraints. A moderate GAV mapping contains twenty GAV constraints in total: six `COPY`, six `DelAttr`, and eight `Merge` constraints. Finally, a complex GAV mapping contains thirty GAV constraints in total: nine `COPY`, nine `DelAttr`, and twelve `Merge` constraints. To create non-trivial GAV mappings, we set both `SourceShare` and `TargetShare` to 100%[4]. We left `JoinKind` with its default value, namely, star join. We set different `JoinSize` values for each of the three types of mappings. Concretely, `JoinSize` = 3 for the simple type, `JoinSize` = 6 for the moderate type, and `JoinSize` = 9 for the complex type.

Table 1: Statistics of different mapping types

	Copy	DelAttr	Merge	JoinSize	Avg. arity
simple	3	3	4	3	16.7
moderate	6	6	8	6	12.3
complex	9	9	12	9	6.9

For each mapping type, we create 5 random GAV mappings. For each GAV mapping, we consider different combinations of α and ExNum, where α ={0.1,0.3,0.5} and ExNum={10,30,50,70,90}

Data example generation. One can also ask iBench to produce a random source instance for a schema mapping $\mathcal{G}$ by assigning an integer value to the `NumOfElem` parameter. If `NumOfElem` = n, then the source instance produced by iBench has n facts per source relation. To avoid such uniformity, we introduce a sampling parameter α, where $0 < \alpha < 1$, to create a random sub-instance from the raw instance produced by iBench. Fix a schema mapping $\mathcal{G}$ and a sampling ratio α. If $I = \{f_1, \ldots, f_m\}$ is a source instance for $\mathcal{G}$ produced by iBench, we then produce a sub-instance I' from I such that every fact $f_k \in I$ has α chance to be out into I'. Once we have a source instance I' created this way, we can obtain the canonical universal solution for I' w.r.t. $\mathcal{G}$ using the chase procedure described in [20], which in turn gives rise to a universal example for $\mathcal{G}$. To create multiple random universal examples, we simply repeat the above process multiple times. Thus, the number of universal examples, denoted by `ExNum`, is another tunable parameter in our evaluation.

Evaluation Methodology. Our goal was to evaluate the performance of GAVLearn on learning GAV mappings of the three different types. To amortize the influence of possible outliers, we created ten random GAV mappings for each type and measured the average recall, F-score, and runtime of GAVLearn with respect to the ten mappings of each type. Our evaluation methodology is summarized as follows.

Fix a type X (simple, moderate, or complex) and a GAV mapping of type X. For each combination $\langle \alpha, \texttt{ExNum} \rangle$, with $\alpha \in \{0.1, 0.3, 0.5\}$ and $\texttt{ExNum} \in \{10, 30, 50, 70, 90\}$, we do the following:

(1) Generate a set S consisting of `ExNum` examples using the example generation procedure described above (with parameters `NumOfElem` = 5 and α).[5]
(2) Split S into a training set E (50%) and a test set V (50%).
(3) Run GAVLearn on E to get a mapping $\mathcal{H}$ and evaluate $\mathcal{H}$ on V.

When evaluating $\mathcal{H}$ on V, we compute the average performance (e.g., recall) of $\mathcal{H}$ achieved on all examples in V. Since we have five random mappings of each type, where each mapping has a corresponding learned mapping $\mathcal{H}$, the numbers reported in Section 4.1 is the average performance of these ten $\mathcal{H}$'s achieved on the corresponding V's.

An overview of the statistics of the three mapping types is provided in Table 1.

Implementation. GAVLearn was implemented in Java. We used PostgreSQL v9.3.10 with default settings. All experiments were run on a 64-bit Linux machine with a Intel i7-4770 CPU (3.40GHz) and 20GB RAM. Our implementation of GAVLearn and the data examples used in the experiments can be found online[6].

[4]We observed that the actual percentage of shared source and target relations in generated mappings is around 30%

[5]We observed that the number of facts in the source instances we generated ranges from ~ 5 to ~ 400

[6]https://github.com/qk63030898/GAVLearn

Table 2: Results of GAVLearn on simple type

α	n	$\lvert E\rvert$	$\overline{Comp}$	$\overline{Rep}$	$\overline{Recall}$	F_s	$\overline{Time}$
0.1	10	5	0.52±8%	0.83±20%	0.832±18%	0.907	0.3s
	30	15	0.6±0%	0.91±7%	0.984±1%	0.992	0.7s
	50	25	0.58±16%	0.97±3%	0.998±13%	0.999	1.1s
	70	35	0.66±5%	0.92±5%	0.992±1%	0.996	0.7s
	90	45	0.7±7%	0.89±15%	0.992±1%	0.995	0.7s
0.3	10	5	0.74±31%	0.95±6%	0.988±9%	0.993	2.2s
	30	15	0.88±8%	0.89±7%	0.982±2%	0.991	2.4s
	50	25	0.96±8%	0.96±8%	0.992±1%	0.995	2.3s
	70	35	1	1	1	1	4.2s
	90	45					2.2s
0.5	10	5	0.98±4%	0.97±4%	0.992±1%	0.996	3.4s
	30	35					3.8s
	50	45	1	1	1	1	3.6s
	70	45					4.1s
	90	45					3.8s

Table 3: Results of GAVLearn on moderate type

α	n	$\lvert E\rvert$	$\overline{Comp}$	$\overline{Rep}$	$\overline{Recall}$	F_s	$\overline{Time}$
0.1	10	5	0.59±22%	0.98±4%	0.99±1%	0.996	4.4s
	30	15					4.9s
	50	25	0.60	1	1	1	4.2s
	70	35					4.9s
	90	45					5.4s
0.3	10	5	0.61±2%	0.98±3%	0.998±4‰	0.998	15.2s
	30	15	0.61±2%	1	1	1	16.2s
	50	25	0.65±3%	0.95±6%	0.998±1‰	0.998	25.6s
	70	35	0.63±2%	0.92±5%	0.997±2‰	0.997	13.6s
	90	45	0.65±3%	0.85±4%	0.997±1‰	0.998	18.2s
0.5	10	5	0.72±2%	0.88±8%	0.985±1‰	0.992	28.4s
	30	15	0.89±4%	0.90±3%	0.992±4‰	0.996	38.3s
	50	25	0.92±5%	0.92±5%	0.995±3‰	0.997	39s
	70	35	0.95±5%	0.95±4%	0.997±3‰	0.998	42s
	90	45	1	1	1	1	50s

Table 4: Results of GAVLearn on complex type

α	n	$\lvert E\rvert$	$\overline{Comp}$	$\overline{Rep}$	$\overline{Recall}$	F_s	$\overline{\text{Time}}$
0.1	10	5	0.53±5%	0.87±10%	0.882±10%	0.937	17.8s
	30	15					20.7s
	50	25	0.60±0%	1	1	1	17.6s
	70	35					22.4s
	90	45					19.7s
0.3	10	5	0.60±0%				1m26s
	30	15	0.60±1%	1	1	1	1m35s
	50	25	0.60±0%				1m34s
	70	35	0.60±1%				1m33s
	90	45	0.60±0%	0.98±2%	0.999	0.999	1m45s
0.5	10	5	0.63±3%	0.98±2%	0.998±4‰	0.999	4m15s
	30	15	0.64±4%	0.92±5%	0.997±2‰	0.998	4m43s
	50	25	0.69±3%	0.92±4%	0.998±1%	0.999	6m55s
	70	35	0.73±4%	0.87±9%	0.998±1%	0.999	5m44s
	90	45	0.74±2%	0.88±8%	0.998±1%	0.999	10m9s
0.7	10	5	0.81±5%	0.84±3%	0.98±7‰	0.99	13m2s
	50	25	1	1	0.999±1‰	0.999	15m5s

Comprehensiveness and Representativeness. To shed more light on the experimental results, we introduce two additional measures: (i) the *comprehensiveness* of a training set E with respect to a goal mapping $\mathcal{G}$, denoted by $Comp_{\mathcal{G}}(E)$; (ii) the *representativeness* of a training set E with respect to a goal mapping $\mathcal{G}$ and test set V, denoted by $Rep_{\mathcal{G},V}(E)$.

Let $\mathcal{G} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a goal GAV schema mapping in a learning task, where Σ is a set of GAV constraints. Let E and V be, respectively, a training set and a test set created for $\mathcal{G}$. We define

$$Comp_{\mathcal{G}}(E) = \frac{|\Sigma^{\mathrm{E}}|}{|\Sigma|} \text{ and } Rep_{\mathcal{G},V}(E) = \frac{|\Sigma^{\mathrm{E}} \cap \Sigma^{\mathrm{V}}|}{|\Sigma^{\mathrm{V}}|},$$

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). Both $Comp_{\mathcal{G}}(E)$ and $Rep_{\mathcal{G},V}(E)$ impact the outcome of a learning task. Intuitively, a high $Rep_{\mathcal{G},V}(E)$ value indicates that the training set E provides the learning algorithm with sufficient information to learn a high quality (or even perfect) schema mapping. Moreover, a low $Rep_{\mathcal{G},V}(E)$ value indicates that, most likely, any learning algorithm would have poor result because the hold-out test contains a richer set of examples than the training set does. The measure $Comp_{\mathcal{G}}(E)$ indicates whether the training set E covers different kinds of GAV constraints; this aims to test how well the learning algorithm can deal with different kinds of constraints.

4.1 Stand-alone Evaluation of GAVLearn

Highlights of the results of GAVLearn are provided in Tables 2, 3, and 4. Each row of the three tables corresponds to a particular learning task. In what follows, we use the expression "X-$\alpha(\theta)$-$n(\beta)$ " to refer to the learning task in which the type of goal schema mappings in consideration is X, the sampling parameter α has value θ, and the number n = ExNum of universal examples is β. Since we used a 50/50 split, the size $|E|$ of the training set E is $n/2$. For example, Complex-$\alpha(0.1)$-$n(10)$ refers to the first row of Table 4, thus it is the learning task for complex type schema mappings with the sampling parameter equal to 0.1 and the number of universal examples equal to 10.

We report the following quantities: the average $Comp_{\mathcal{G}}(E)$ (denoted by $\overline{Comp}$), the average $Rep_{\mathcal{G},V}(E)$ (denoted by $\overline{Rep}$), the average recall (denoted by $\overline{Recall}$), the overall F-score (denoted by F_s), and the average runtime (denoted by $\overline{Time}$) of GAVLearn. We also report the standard deviation values of most measures. We do not report the precision of GAVLearn, because it is 100%.

Analysis of Results. As shown in Tables 2, 3, and 4, there is an evident dependence of recall values on the corresponding $\overline{Rep}$ values. In fact, in all tasks, the recall values are quite close to the corresponding $\overline{Rep}$ values. Unlike the $\overline{Rep}$ values that are strongly correlated to recall, the $\overline{Comp}$ values are less relevant to recall. This observation is consistent with our expectation that the $\overline{Rep}$ value of a training set has a significant influence on recall, while the $\overline{Comp}$ value is just an indicator of the "difficulty" of a learning task. The similarity between $\overline{Rep}$ values and recall shows that GAVLearn is able to acquire the information provided by training sets to produce GAV mappings that are very close to the best solutions any learning algorithm can produce.

Although the $\overline{Comp}$ values are not strongly correlated to recall, the high recall values achieved by GAVLearn are more meaningful if the corresponding $\overline{Comp}$ values are also high (a training set of high $\overline{Comp}$ value covers a variety of constraints, including the more sophisticated constraints defined by the merge primitive). To challenge GAVLearn, we further increased α to 0.7 in the case of complex type, so that the resulting training sets can cover a significant fraction of the constraints of the goal mappings. As stated earlier, each GAV mapping of complex type has thirty GAV constraints, twelve of which are rather sophisticated GAV constraints (each has nine relational atoms joining on the left-hand side). The numbers in the bottom of Table 4 show that GAVLearn can successfully learn those complex GAV mappings with very high recall.

Observe that, in most cases, when $\overline{Rep} = 1$, then also $\overline{Recall} = 1$. In our experiments, we observe a couple of exceptions in which the recal values are not 1 when $\overline{Rep} = 1$ (e.g., Complex-$\alpha(0.7)$-$n(50)$). The main reason for these exceptions is that, during the learning process, GAVLearn computes only one critically sound constraint from a given candidate constraint C, even if there are more than

one critically sound constraints of C. This is a heuristic that we implemented in GAVLearn to improve the efficiency of the learning algorithm, but it may result in a loss of recall.

As an illustration, let $\mathcal{M}$ be the goal mapping consisting of the constraints: $P(x) \rightarrow Q(x)$ and $M(x) \rightarrow Q(x)$. Consider the training example (I, J), where $I = \{P(a), M(a)\}$ and $J = \{Q(a)\}$. This example triggers the two GAV constraints of $\mathcal{M}$, so both $\overline{Comp}$ and $\overline{Rep}$ are 1. When we run GAVLearn on (I, J), either $P(x) \rightarrow Q(x)$ or $M(x) \rightarrow Q(x)$ will be generated, but not both, because of our one-critically-sound-constraint heuristic. If GAVLearn generates $P(x) \rightarrow Q(x)$, then the resulting schema mapping will achieve zero recall on the test example (I', J'), where $I' = \{M(a)\}$ and $J' = \{Q(a)\}$. GAVLearn can be extended to generate all possible critically sound constraints, but that would result in an exponential running time.

Next, we discuss two other aspects of GAVLearn.

Critically Sound Constraints One of the advantages of GAVLearn is that it systematically generates a large number of additional universal examples from the given training set. These additional universal examples are important because they lead GAVLearn to create critically sound constraints w.r.t. the goal mapping, which according to Occam's Razor Principle are likely to generalize well on future data. By Lemma 10, 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., in the total number of facts contained in E). For example, the mapping returned by GAVLearn, in the `Complex`-$\alpha(0.5)$-$n(90)$ task, was actually an outcome of learning from $12K \approx 45 \times 280$ universal examples. These additional examples contributed significantly to the high-quality result.

Runtime efficiency The runtime of GAVLearn (reported in the last column of each table) depends mainly on the size of the training set. Because of the computation of critically sound constraints, a very large training set would cause a bottleneck. However, our experiments were designed to simulate a common user-interactive schema-mapping design environment, where the number of examples is usually comparable to the size of constraints of the goal schema mapping. Thus, the runtime of GAVLearn is reasonable.

4.2 Comparison with Other Approaches

In this section, we compare GAVLearn with two earlier approaches: the GAV-Fit algorithm [1] and the TALOS system. These two baselines were run on the same sets of training data and test data that were used for evaluating GAVLearn. Figure 2 depicts the F-score values achieved by the three methods in different learning tasks. It is clear that GAVLearn performed much better than the two baselines, while TALOS produced better results than GAV-Fit. We now discuss the differences among them in detail.

GAV-Fit. Introduced in [1], GAV-Fit is an algorithm that, given a set E of ground data examples, generates the *canonical GAV fitting schema mapping* for E if such a fitting GAV mapping for E exists. By definition, the canonical GAV schema mapping of E is obtained by extracting a GAV constraint from each data example (I, J) in E, so that the left-hand side of the constraint is the canonical database of I and the right-hand side is the canonical database of J. In view of these characteristics, the canonical GAV schema mapping does not generalize well on the hold-out test set V. In our evaluation, the canonical GAV mapping returned by GAV-Fit achieved high precision, but low recall on V (most recall values are zero), because the canonical GAV mapping obtained from E would produce a non-empty target instance only if one of the source instances of V contains a homomorphic image of one of the source instances of E. From a machine learning point of view, GAV-Fit tends to over-fit, and this accounts for its low F-scores.

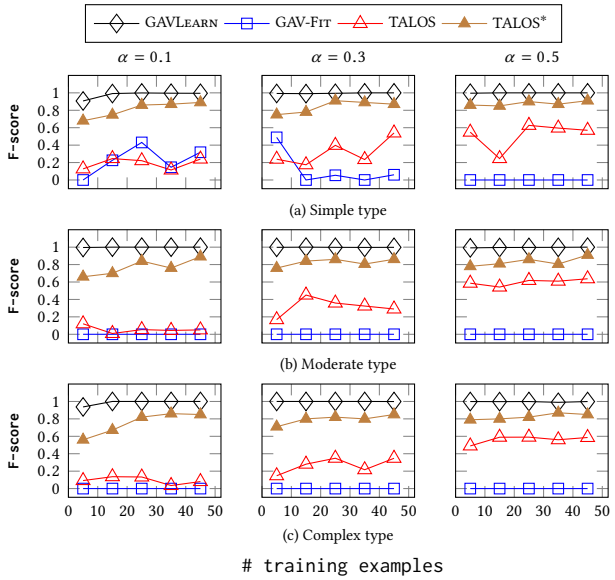

Figure 2: Comparison of GAV-Fit, TALOS, and GAVLearn

TALOS. The TALOS system[7] was originally designed for reverse engineering of unions of conjunctive queries. It is easy to see that there is a tight correspondence between GAV mappings and unions of conjunctive queries; in particular, the right-hand side of a GAV constraint can be viewed as the result of the conjunctive query over the source formed by the left-hand side of the GAV constraint.

Figure 2 shows that TALOS performed much better than GAV-Fit. The core technique of TALOS is a classification algorithm based on a decision tree. This algorithm re-organizes the constants of the input database by a decision tree such that multiple instance-equivalent SPJU queries (and, hence, GAV mappings) can be extracted from the decision tree with respect to the query result provided. When constructing a GAV mapping, TALOS will return one of minimum cost among all candidate mappings (see [37] for details). One of the advantages of TALOS over GAV-Fit is that TALOS takes into account the "succinctness" of a learned model. For this reason, it will return a model that is less specific to the input database (the training set in our setting).

By manually inspecting the mappings returned by TALOS, we found that a substantial portion of these mappings contain constants (used in equality conditions in the resulting constraints) belonging to the training set E. This means that TALOS still suffers from some amount of over-fitting. As regards to runtime efficiency, TALOS is highly efficient (all tests finished within two seconds) mainly because it considers only the given training set.

[7] The authors of [37] kindly shared with us the implementation of the TALOS system.

4.3 Impact of Elicited Universal Examples

As already discussed earlier, the key difference between GAVLearn and the two baselines is that GAVLearn is able to actively query the provided oracle to generated more informative universal examples. These additional universal examples are crucial for GAVLearn, as these examples guide the algorithm toward GAV constraints that are semantically close to the GAV constraints of the goal GAV mapping. In fact, we can show empirically that the universal examples generated by GAVLearn are highly informative in the sense that, not only they are beneficial for GAVLearn, but also they are beneficial for other schema-mapping discovery algorithms, such as the two considered baselines.

To validate this claim, we carried out one more experiment in which we gave the universal examples collected from the runs of GAVLearn to the two baseline approaches. The results are shown in Figure 2, where TALOS* represents the F-scores of TALOS by using the initial input set of universal examples together with the additional universal examples obtained from a corresponding run of GAVLearn. Note that Figure 2 does not plot the F-score values of GAV-Fit that uses the additional universal examples. This is because the F-scores of GAV-Fit coincide with the F-scores of GAVLearn. In fact, we have the following proposition.

Proposition 21. *Let E be the set of universal examples for a GAV mapping $\mathcal{G}$. Let $\mathcal{H}$ be the GAV mapping obtained by GAVLearn on input E and with oracle access to $\mathcal{G}$, and, moreover, let E' be the set of all pairs $(I, \texttt{can-sol}_{\mathcal{G}}(I))$, where I is a query to the oracle during the run of the algorithm. Then $\mathcal{H}$ is logically equivalent to the GAV mapping $\mathcal{H}$' produced by running GAV-Fit over $E \cup E'$.*

The proof proceeds by showing that $\mathcal{H}$ is a *most-general fitting GAV mapping* for the set of universal examples $E \cup E'$. The concept of a most-general fitting GAV mapping was introduced in [1], where it was shown to be unique up to logical equivalence, and where it was also shown that GAV-Fit produces a most-general fitting schema mapping for its input set of universal examples.

In particular, Proposition 21 shows that GAVLearn and GAV-Fit obtain the same F-score when the input to GAV-Fit is augmented with the additional examples elicited by GAVLearn. As regards TALOS, its performance also improved significantly by giving it the elicited universal examples E'; indeed, as seen in Figure 2, the F-scores of TALOS * are much higher than those of TALOS. However, the performance of TALOS * does not match that of GAVLearn. As discussed earlier, the mappings returned by TALOS still use some constants of the input data examples, so it still suffers from over-fitting issues.

5 OTHER RELATED WORK

Recently, Kimmig et al. [27] used a probabilistic model for schema-mapping discovery. The input is a set of data examples and a set of candidate constraints, and the objective is to form a schema mapping by selecting a suitable subset of the input set of constraints that satisfies certain criteria. Clearly, this is a different approach from ours, since it requires as input a set of data examples and a set of constraints, while GAVLearn requires a set of data examples and a labeling oracle for the unknown goal mapping.

There is a body of work on learning queries from examples [14, 15, 32, 35, 37, 40]. Since there is a correspondence between GAV mappings and union of conjunctive queries, methods for learning queries can be used, in principle, to learn GAV mappings. In [14, 15, 32], active learning based methods have been proposed to learn queries. In these approaches, the learning involves a human user who labels "informative" tuples as positive or negative. The main assumption of these approaches is that a perfect query can always be derived by learning from user labels, hence the goal is to minimize the number of user interactions. In these systems, the human user corresponds to an oracle that gives exact answers to equivalence queries. In contrast, our approach focuses on approximately learning complex GAV mappings (which are very hard for a user to understand) using a stochastic equivalence oracle. In [35], the goal is to discover a minimal project-join query. For this, the user has to pick one of the candidate queries produced by the algorithm and to also modify the query picked to obtain the final goal query. In [37, 40], queries are derived using knowledge about integrity constraints. The following problem was studied: given a database D and given a table T that is the result of some unknown SPJ query Q on D, derive a query Q' such that the result of Q' on D is equal to T. Here, we compared GAVLearn with the TALOS system introduced in [37] since it supports reverse engineering unions of conjunctive queries.

6 CONCLUDING REMARKS

The work reported here demonstrates that a theoretical result can give rise to a practical tool. Specifically, building on the theoretical results in [17] about the exact learnability of GAV mappings with labeling and equivalence oracles, we designed, analyzed, and experimentally evaluated GAVLearn, an active, example-driven learning algorithm for discovering GAV mappings. The experiments showed that GAVLearn can effectively learn high-quality GAV mappings from universal examples; moreover, it outperforms two other algorithms, namely, GAV-Fit and TALOS. The key difference between GAVLearn and the other two algorithms is that GAVLearn uses the provided black-box goal mapping to generate additional universal examples that lead the algorithm to create critically sound constraints. Both GAV-Fit and TALOS focus on producing a GAV mapping that perfectly fits only the training set. This, in some sense, explains the superior performance of GAVLearn.

Several other important classes of schema mappings merit investigation using the learnability lens. In particular, the learnability of LAV (local-as-view) mappings remains an open problem at both the theoretical level (e.g., are LAV mappings exactly learnable using labeling queries and equivalence queries?) and at the practical level (are there good active learning algorithms for discovering LAV mappings?). The same questions remain unanswered for the class of GLAV mappings and for the broader class of schema mappings specified by SO-tgds (second-order tuple-generating dependencies), which express the composition of GLAV mappings [22].

So far, none of the different frameworks for schema-mapping discovery has considered schema mappings whose specification includes target constraints. For such schema mappings, it is meaningful to consider different notions of equivalence, such as *conjunctive-query equivalence* [21], and to investigate their learnability from this perspective.

REFERENCES

[1] B. Alexe, B. ten Cate, P. Kolaitis, and W.-T. Tan. 2011. Designing and refining schema mappings via data examples. In *SIGMOD*. 133–144.

[2] B. Alexe, B. ten Cate, P. Kolaitis, and W.-T. Tan. 2011. EIRENE: Interactive Design and Refinement of Schema Mappings via Data Examples. *PVLDB* (2011), 1414–1417.

[3] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. 2011. Characterizing Schema Mappings via Data Examples. *ACM Trans. Database Syst.* 36, 4, Article 23 (Dec. 2011), 48 pages. https://doi.org/10.1145/2043652.2043656

[4] Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (1987), 87–106. https://doi.org/10.1016/0890-5401(87)90052-6

[5] D. Angluin. 1988. Queries and Concept Learning. *Machine Learning* (1988), 319–342.

[6] Dana Angluin. 1990. Negative Results for Equivalence Queries. *Machine Learning* 5 (1990), 121–150. https://doi.org/10.1007/BF00116034

[7] Dana Angluin, Timos Antonopoulos, and Dana Fisman. 2017. Query Learning of Derived Omega-Tree Languages in Polynomial Time. In *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden (LIPIcs)*, Valentin Goranko and Mads Dam (Eds.), Vol. 82. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 10:1–10:21. https://doi.org/10.4230/LIPIcs.CSL.2017.10

[8] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. 2014. *Foundations of Data Exchange*. Cambridge University Press.

[9] Marta Arias, Roni Khardon, JÃľrÃťme Maloberti, and Stefan Wrobel. 2007. Learning Horn Expressions with LOGAN-H. (2007), 549-587 pages.

[10] P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller. 2015. The iBench Integration Metadata Generator. *PVLDB* (2015), 108–119.

[11] P. Barceló. 2009. Logical foundations of relational data exchange. *SIGMOD* (2009), 49–58.

[12] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. 1987. Occam's Razor. *Inf. Process. Lett.* (1987), 377–380.

[13] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger. 2005. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *VLDB*. 1267–1270.

[14] A. Bonifati, R. Ciucanu, and S. Staworko. 2014. Interactive Join Query Inference with JIM. *VLDB*. (2014), 1541–1544.

[15] A. Bonifati, R. Ciucanu, and S. Staworko. 2016. Learning Join Queries from User Examples. *ACM TODS* (2016), 24:1–24:38.

[16] Angela Bonifati, Ugo Comignani, Emmanuel Coquery, and Romuald Thion. 2017. Interactive Mapping Specification with Exemplar Tuples. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 667–682. https://doi.org/10.1145/3035918.3064028

[17] B. ten Cate, V. Dalmau, and P. Kolaitis. 2013. Learning Schema Mappings. *TODS* (2013), 28:1–28:31.

[18] B. ten Cate, P. Kolaitis, K. Qian, and W.-C. Tan. 2017. Approximation Algorithms for Schema-Mapping Discovery from Data Examples. *TODS* (2017), 12:1–12:41.

[19] A. K. Chandra and P. M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*. 77–90.

[20] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. 2003. Data Exchange: Semantics and Query Answering. 207–224.

[21] Ronald Fagin, Phokion G. Kolaitis, Alan Nash, and Lucian Popa. 2008. Towards a theory of schema-mapping optimization. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*, Maurizio Lenzerini and Domenico Lembo (Eds.). ACM, 33–42. https://doi.org/10.1145/1376916.1376922

[22] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. 2005. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.* 30, 4 (2005), 994–1055. https://doi.org/10.1145/1114244.1114249

[23] Georg Gottlob, Reinhard Pichler, and Vadim Savenkov. 2009. Normalization and Optimization of Schema Mappings. *PVLDB* 2, 1 (2009), 1102–1113. http://www.vldb.org/pvldb/2/vldb09-923.pdf

[24] G. Gottlob and P. Senellart. 2010. Schema mapping discovery from data instances. *JACM* (2010).

[25] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. 2005. Clio Grows Up: From Research Prototype to Industrial Tool. In *SIGMOD*. 805–810.

[26] Roni Khardon. 1999. Learning Function-Free Horn Expressions. *Machine Learning* 37, 3 (01 Dec 1999), 241–275. https://doi.org/10.1023/A:1007610422992

[27] A. Kimmig, A. Memory, R. J Miller, and L. Getoor. 2017. A collective, probabilistic approach to schema mapping. *ICDE* (2017).

[28] P. Kolaitis. 2005. Schema Mappings, Data Exchange, and Metadata Management.. In *PODS*. 61–75.

[29] Boris Konev, Carsten Lutz, Ana Ozaki, and Frank Wolter. 2017. Exact Learning of Lightweight Description Logic Ontologies. *CoRR* abs/1709.07314 (2017). arXiv:1709.07314 http://arxiv.org/abs/1709.07314

[30] Boris Konev, Ana Ozaki, and Frank Wolter. 2016. A Model for Learning Description Logic Ontologies Based on Exact Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, 1008–1015. http://dl.acm.org/citation.cfm?id=3015812.3015962

[31] M. Lenzerini. 2002. Data Integration: A Theoretical Perspective. In *PODS*. 233–246.

[32] H. Li, C.-Y. Chan, and D. Maier. 2015. Query from Examples: An Iterative, Data-driven Approach to Query Construction. *VLDB* (2015), 2158–2169.

[33] R. J. Miller, L. M. Haas, and M. A. Hernández. 2000. Schema Mapping as Query Discovery. 77–88.

[34] Doron A. Peled, Moshe Y. Vardi, and Mihalis Yannakakis. 2002. Black Box Checking. *Journal of Automata, Languages and Combinatorics* 7, 2 (2002), 225–246.

[35] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. 2014. Discovering Queries Based on Example Tuples. In *SIGMOD*. 493–504.

[36] Bernhard Steffen, Falk Howar, and Maik Merten. 2011. Introduction to Active Automata Learning from a Practical Perspective. In *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures (Lecture Notes in Computer Science)*, Marco Bernardo and Valérie Issarny (Eds.), Vol. 6659. Springer, 256–296. https://doi.org/10.1007/978-3-642-21455-4_8

[37] Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. 2014. Query Reverse Engineering. *The VLDB Journal* (2014), 721–746.

[38] Frits W. Vaandrager. 2017. Model learning. *Commun. ACM* 60, 2 (2017), 86–95. https://doi.org/10.1145/2967606

[39] Leslie G. Valiant. 1984. A Theory of the Learnable. *Commun. ACM* 27, 11 (1984), 1134–1142. https://doi.org/10.1145/1968.1972

[40] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. 2013. Reverse Engineering Complex Join Queries. In *SIGMOD*.

7 APPENDIX

PROOFS

Claim 14. *For every $i \geq 0$ such that $\mathcal{H}_i$ is defined, $\mathcal{H}_i$ consists of GAV constraints that are sound with respect to $\mathcal{G}$. In particular, in Line 6 of GAVLearn, it must be the case that* $\mathsf{can\text{-}sol}_{\mathcal{H}_i}(I) \subsetneq J$.

Proof. By induction on i. Claim 14 is trivially true for $i = 0$, since $\mathcal{H}_0 = \emptyset$. Now, for the case of $i + 1$, let $(I, J) \in E$ be the data example chosen in Line 6. Note that $J = \mathsf{can\text{-}sol}_{\mathcal{G}}(I)$. By the induction hypothesis, $\mathcal{H}_i$ is logically implied by $\mathcal{G}$, and hence, $\mathsf{can\text{-}sol}_{\mathcal{H}_i}(I) \subseteq \mathsf{can\text{-}sol}_{\mathcal{G}}(I) = J$. Let $f \in J \setminus \mathsf{can\text{-}sol}_{\mathcal{H}_i}(I)$ as chosen in Line 8. It follows from Lemma 9 that $\mathcal{G}$ logically implies the GAV constraint $(I, \{f\})$. In other words, $(I, \{f\})$ is sound with respect to $\mathcal{G}$. It follows by Lemma 11 and Lemma 10 that the constraint added to $\mathcal{H}_{i+1}$ in Line 11 or 13, is also sound with respect to $\mathcal{G}$. We conclude that $\mathcal{G}$ logically implies $\mathcal{H}_{i+1}$. □

Claim 15. *Let $i \geq 0$ such that $\mathcal{H}_i$ is defined. For every $C \in \mathcal{H}_i$ there is a $C^* \in \mathcal{G}$ such that $C^* \mapsto C$. Furthermore, for each $C^* \in \mathcal{G}$, the set $\mathcal{H}_i(C^*) = \{C \in \mathcal{H}_i \mid C^* \mapsto C\}$ is either empty or is a singleton set.*

Proof. The first part follows directly from Claim 14 together with Lemma 9. For the second part of the claim, we proceed by induction on i. The base case, for $i = 0$, is trivial. For $i > 0$, consider any $C^* \in \mathcal{G}$. Let C'_i be the constraint added to $\mathcal{H}$ in the i-th iteration of the algorithm (either in Line 11 or in Line 13). If $C^* \not\mapsto C'_i$, then the result follows immediately from the induction hypothesis. Therefore, in what follows, we will assume that $C^* \mapsto C'_i$. We distinguish two cases: (i) $\mathcal{H}_{i-1}(C^*) = \emptyset$. In this case, C'_i is the only GAV constraint in $\mathcal{H}_i$ into which C^* maps, and hence the result holds trivially; (ii) $\mathcal{H}_{i-1}(C^*) \neq \emptyset$. By the induction hypothesis, there is a unique $C_j \in \mathcal{H}_{i-1}$ such that $C^* \mapsto C_j$. Note that, in this case, we have that $C^* \mapsto C_j \times (I, \{f\})$ (by the properties of direct products), and hence the if-condition in Line 9 of the algorithm is satisfied. Moreover, we can show that C_j is in fact the constraint C

chosen by the algorithm in Line 10. This follows immediately from the induction hypothesis, because C^* homomorphically maps both to C_j and to C'_i, which, in turn, by the properties of direct product, homomorphically maps to C, since $C'_i = Crit_{\mathcal{G}}(C \times (I, \{f\}))$. Since C is explicitly removed from $\mathcal{H}$ in Line 11, it follows that $C^*(\mathcal{H}_i)$ is again a singleton set. This concludes the proof for Claim 15. □

Claim 16. *The size of the schema mapping $\mathcal{H}$ produced by GAVLearn is at most the size of $\mathcal{G}$.*

Proof. By Claim 15 we have that, for each constraint C_i in $\mathcal{H}$ there is a constraint C^*_i in $\mathcal{G}$ such that $C^*_i \mapsto C_i$, and, moreover, for $i \neq j$, we have that $C^*_i \neq C^*_j$. In other words, the constraints of $\mathcal{H}$ stand in a one-to-one correspondence with a subset of the constraints of $\mathcal{G}$. Furthermore, it is easy to see that each homomorphism in question must be surjective, otherwise the constraint C_i would not be *critically* sound with respect to $\mathcal{G}$. Therefore, each constraint C_i is of size at most the size of C^*_i. The claim immediately follows. □

Claim 17. *For every $i > 0$ such that $\mathcal{H}_i$ is defined, we have $s_i > s_{i-1}$, and $s_i \leq n$, where $n = \Sigma_{C \in \mathcal{G}} nvar(C)$.*

Proof. We first show that $s_{i+1} > s_i$. Let C_{i+1} be the constraint added to $\mathcal{H}_{i+1}$ in the $i+1$-th iteration. Since C_{i+1} is critically sound with respect to $\mathcal{G}$, there is a $T \in \mathcal{G}$ such that $T \mapsto C_{i+1}$. By Claim 15, C_{i+1} is the unique element of $\mathcal{H}_{i+1}(T)$. It follows that s^T_{i+1} is the domain size of C_{i+1}.

First, consider the case where $\mathcal{H}_i(T)$ is empty. In this case, C_{i+1} cannot be of the form $Crit_{\mathcal{G}}((I, \{f\}) \times C$ with $C \in \mathcal{H}_i$, because this would imply that $C_{i+1} \mapsto C$ and hence $C \in \mathcal{H}_i(T)$. In other words, C_{i+1} must be of the form $Crit_{\mathcal{G}}((I, \{f\}))$, and must have been added to $\mathcal{H}_{i+1}$ in line 13 of the algorithm. It immediately follows that $s_{i+1} > s_i$.

Next, consider the case where $\mathcal{H}_i(T)$ is non-empty, and let C_i be its unique element. It follows from Claim 15 that C_i must be the constraint removed from $\mathcal{H}_i$ by the algorithm in the i+1-th iteration, and hence $C_{i+1} = Crit_{\mathcal{G}}(C_i \times (I, \{f\}))$ for some I and f. In particular, we have that $C_{i+1} \mapsto C_i$. In fact, the homomorphism h from the left-hand side of C_{i+1} to the left-hand side of C_i must be surjective, in other words, C_i must be the h-image of C_{i+1}, for otherwise, we could obtain a non-surjective homomorphism from the left-hand side of T to the left-hand side of C_i (namely the composition of h with the homomorphism $T \mapsto C_{i+1}$), contradicting, via Lemma 9, the fact that C_i is critically sound with respect to $\mathcal{G}$. We also know that C_{i+1} is not isomorphic to C_i, because $C_{i+1} \mapsto (I, \{f\})$ whereas $C_i \not\mapsto (I, \{f\})$. It follows that the domain size of C_{i+1} is larger than that of C_i (otherwise h would be an isomorphism) and Claim 17 is proved.

Next we show that each s_i is at most n. Assume, for the sake of a contradiction, that $s_i > n$, for some well defined s_i. Then there is a constraint $C \in \mathcal{H}_i$ and $C^* \in \mathcal{G}$ with $\mathcal{H}_i(C^*) = \{C\}$ such that $nvar(C) > nvar(C^*)$. This means that there is a non-surjective homomorphism from C^* to C, resulting in a contradiction to the fact that C is critically sound with respect to $\mathcal{G}$. □

Theorem 20. *GAVLearn asks at most*

$$size(\mathcal{G})^2 \cdot maxsize(E)$$

many labeling queries, where $size(\mathcal{G})$ is the size of the goal schema mapping, and $maxsize(E)$ is the maximal number of facts in an input data example.

Proof. In fact, we show a slightly stronger bound: GAVLearn asks at most

$$n_{\mathcal{G}} \cdot (|\mathcal{G}| + k_{\mathcal{G}} \cdot k_E)$$

many labeling queries, where where $n_{\mathcal{G}} = \Sigma_{C \in \mathcal{G}} nvar(C)$, $|\mathcal{G}|$ is the number of constraints of $\mathcal{G}$, $k_{\mathcal{G}}$ is the maximum number of atoms in the left-hand side of a constraint of $\mathcal{G}$, and $k_E = \max_{(I,J) \in E} |I|$ is the maximum number of source facts of a data example in E.

By Claim 17, the number of iterations of the algorithm is bounded by $n_{\mathcal{G}}$. In each iteration, the labeling oracle is called in Line 9 (for testing logical implication) at most $|\mathcal{G}|$ times, and in Line 11 or 13 (for computing the critically sound subconstraint) at most $k_{\mathcal{G}} \cdot k_E$ times, since the number of atoms in the left hand side of the constraint $(I, \{f\}) \times C$ is bounded by $k_{\mathcal{G}} \cdot k_E$, and at most one labeling query is performed per atom in the left-hand side of the constraint. To see that the number of atoms in the left hand side of the product constraint $(I, \{f\}) \times C$ is bounded by $k_{\mathcal{G}} \cdot k_E$, note that the fact that the left-hand side of the constraint $C \in \mathcal{H}$ cannot contain more atoms than the left-hand side of a constraint in $\mathcal{G}$ (see the proof of Claim 16). □