Schema Independent Relational Learning

Jose Picado **Arash Termehchy** **Alan Fern**
School of EECS, Oregon State University
Corvallis, OR 97331
{picadolj,termehca,afern}@eecs.oregonstate.edu

Abstract

Relational learning algorithms leverage the structure of a relational database to find the definition of the target relation in terms of the existing relations in the database. The same data may be represented under different relational structures, i.e., schemas. Unfortunately, the accuracy of relational learning algorithms tend to vary quite substantially over the choice of schema, both in terms of learning accuracy and efficiency, which complicates their off-the-shelf application. We introduce the property of schema independence of relational learning algorithms and show that current algorithms are not schema independent on some common variations in schema design. We further modify an existing algorithm and prove that the modified version is schema independent.

1 Introduction

Relational learning algorithms attempt to learn concepts directly from a relational database, without requiring the intermediate step of feature engineering [11, 9, 5]. Given a relational database and training instances of a new target relation, relational learning algorithms leverage the structure of the database and induce (approximate) relational definitions of the target relation in terms of existing relations in the database. Nevertheless, people often represent the same data in different relational structures, i.e., *schemas* [1, 12]. As an example, Table 1 shows some relations from two schemas for the UW-CSE database, which is used as a common relational learning benchmark [13]. The original schema was designed by relational learning experts. However, this schema is generally discouraged in database community because the database system has to spend a long time and join a relatively large number of relations to answer most queries over the databases with this schema [12, 3, 1]. This schema is also relatively hard to understand for users. A database designer may use schemas closer to the alternative in Table 1. Because each student *stud* has only one *phase* and *years*, she will compose relations *student*, *inPhase*, and *yearsInProgram*. She may also combine relations *professor* and *hasPosition*. This would result in shorter query execution time and easier to understand and maintain schema [12]. Note that restructuring the UW-CSE database from the original to alternative schema does not remove, modify, or add any data item in the database and only changes their organization.

Unfortunately, the output of relational learning algorithms typically depend on the precise choice of schema of the underlying database. For instance, let us use a classic relational learning algorithm FOIL [11] to induce a definition of $advisedBy(stud, prof)$ for each of the two structures of UW-CSE data set in Table 1. FOIL learns the following definition over the original schema on Table 1:

$$advisedBy(X, Y) \leftarrow inPhase(X, \text{“}post\_generals\text{”}), hasPosition(Y, \text{“}faculty\text{”}),$$
$$publication(P, X), publication(P, Y).,$$

which covers 23 positive examples and 3 negative examples on the testing set. On the other hand, FOIL learns the following definition over the alternative schema: $advisedBy(X, Y) \leftarrow false.$, which covers 0 positive example and 32 negative examples. Intuitively, the definition learned over the original schema better expresses the relationship between an advisor and advisee. Note that these definitions are learned by FOIL with the exact same data under different schemas.

Original Schema			Alternative Schema	
student(stud)	publication(title,person)	professor(prof)	student(stud,phase,years)	courseLevel(crs,level)
inPhase(stud,phase)	courseLevel(crs,level)	hasPosition(prof,pos)	professor(prof,pos)	taughtBy(crs,prof,term)
yearsInProgram(stud,years)	taughtBy(crs,prof,term)	ta(crs,stud,term)	publication(title,person)	ta(crs,stud,term)

Table 1: Fragments of some schemas for UW-CSE data set. Primary key attributes are underlined.

Generally, there is no canonical schema for a particular set of content in practice and people often represent the same information in different schemas [1]. People may choose to represent their data in one schema or another for several reasons. For example, it is generally easier to enforce integrity constraints over highly normalized schemas [1]. On the other hand, because more normalized schemas usually contain many relations, they are hard to understand and maintain. It also takes a relatively long time to answer queries over database instances with such schemas [1]. Thus, a database designer may sacrifice data quality and choose a more *denormalized* schema for its data to achieve better usability and/or performance. Further, as the relative priorities of these objectives change over time, the schema will also evolve.

In order to effectively use relational learning algorithms, i.e., deliver definitions for the target concepts that a domain expert would judge as correct and relevant, users generally have to restructure their databases to some *proper schema*. To make matters worse, these algorithms do not normally offer any clear description of their desired schema and users have to rely on their own expertise to find such schemas. Nevertheless, we ideally want our database analytics algorithms to be used by ordinary users, not just experts. These users should not know the schema in which the data is represented or tune the algorithm depending on the schema. Further, the structure of large-scale databases constantly evolves, and we want to move away from the need for constant expert attention to keep learning algorithms effective. One approach to solving the problem of schema dependence is to run a learning algorithm over *all possible schemas* select the schema with the most accurate answers. Nonetheless, computing all possible schemas of a DB is generally undecidable [4]. Even if one limits the search space to a particular family of schemas, the number of possible schemas may be extremely large [1, 12].

In this paper, we introduce the novel property of *schema independence* i.e., the ability to deliver the same answers regardless of the choices of schema for the same data, for relational learning algorithms. We propose a formal framework to measure the amount of schema independence of a relational learning algorithm. Since none of the current algorithms are schema independent, we leverage concepts from database literature to extend a relational learning algorithm, ProGolem [9], and design a schema independent algorithm. Further details on our work and proofs for our theoretical results can be found in [10].

2 Background

Let $Attr$ be a set of symbols that contains the names of *attributes* [1]. Each relation R is a subset of $Attr$. Let D be a countably infinite domain of values, e.g., string. A *relation instance* I_R of R assigns a finite relation $I_R \subset D^k$ to R with arity k. A *schema* $\mathcal{R}$ is a pair $(\mathbf{R}, \Sigma)$, where $\mathbf{R}$ is a finite set of relations and Σ is a set of (logical) constraints. Each instance of $\mathcal{R}$, $I_\mathcal{R}$, is a set of instances of relations in $\mathbf{R}$ that satisfy Σ. Examples of constraints are *functional dependencies* (FD) and *inclusion dependencies* (IND). FD $A \rightarrow B$ in relation R, where $A, B \subset R$, states that the values of attribute set A uniquely determine the values of attributes in B in each tuple in every relation instance I_R. For example, FD $stud \rightarrow phase$ holds in relations $inPhase$ of the original schema and $student$ of the alternative schema in Table 1. An IND between attribute C in relation R_1 and D in relation R_2, denoted as $R_1[C] \subseteq R_2[D]$, states that in all instances of I_{R_1} and I_{R_2}, values of attribute C in any tuple of I_{R_1} must also appear in attribute D of some tuple of I_{R_2}. For example, INDs $student[stud] \subseteq inPhase[stud]$ and $inPhase[stud] \subseteq student[stud]$ hold in the original schema of Table 1. If both INDs $R_1[C] \subseteq R_2[D]$ and $R_2[D] \subseteq R_1[C]$ are in Σ, we denote them as $R_1[C] = R_2[D]$ for brevity. We call such IND an IND with equality. Given relation R_i, we call the set of all relations $R_j \in \mathbf{R}$ such that there is an IND $R_i[C] = R_j[D]$ in Σ, the *inclusion class* of R_i.

We denote the set of all Horn definitions over schema $\mathcal{R}$ by $\mathcal{HD}_\mathcal{R}$. A relational learning algorithm takes as input training data E and instance I and learns a hypothesis, i.e., definition, that, together with I, entails E. Most relational learning algorithms limit their hypotheses to Horn definitions. An algorithm may even restrict its hypothesis space to a subset of Horn definitions for various reasons, such as efficiency. We denote the hypothesis space of relational algorithm A over schema $\mathcal{R}$ as

$\mathcal{L}^A_\mathcal{R}$. We define a relational learning algorithm A as a function $A(I, E)$ that maps pairs of database instance I and training data E to a hypothesis in $\mathcal{L}^A_\mathcal{R}$.

3 Framework

Intuitively, in order to learn semantically equivalent definitions over schemas $\mathcal{R}$ and $\mathcal{S}$, we should make sure $\mathcal{R}$ and $\mathcal{S}$ contain basically the same information. Transformation $\tau : \mathcal{R} \rightarrow \mathcal{S}$ maps each instance of $\mathcal{R}$ to an instance of $\mathcal{S}$. If τ is *bijective*, then one is able to construct each instance in $\mathcal{S}$ using the information in its corresponding instance in $\mathcal{R}$ and reconstruct the instance in $\mathcal{R}$ using its corresponding instance in $\mathcal{S}$. Hence, $\mathcal{R}$ and $\mathcal{S}$ represent the same information [6, 1]. Clearly, if τ is bijective, τ^{-1} is also bijective.

We should also make sure that for every definition $h_\mathcal{R} \in \mathcal{HD}_\mathcal{R}$, there is a semantically equivalent definition in $\mathcal{HD}_\mathcal{S}$, and vice versa. Otherwise, it is not reasonable to expect a learning algorithm to learn semantically equivalent definitions over $\mathcal{R}$ and $\mathcal{S}$. Let $h_\mathcal{R} \in \mathcal{HD}_\mathcal{R}$ be a definition over schema $\mathcal{R}$. We denote the result of applying $h_\mathcal{R}$ over instance $I_\mathcal{R}$ as $h_\mathcal{R}(I_\mathcal{R})$. Transformation $\tau : \mathcal{R} \rightarrow \mathcal{S}$ is *definition preserving* iff there exists a total function $\delta_\tau : \mathcal{HD}_\mathcal{R} \rightarrow \mathcal{HD}_\mathcal{S}$ such that for every definition $h_\mathcal{R} \in \mathcal{HD}_\mathcal{R}$ and $I_\mathcal{R}$, $h_\mathcal{R}(I_\mathcal{R}) = \delta_\tau(h_\mathcal{R})(\tau(I_\mathcal{R}))$. In other words, one can rewrite each definition over $\mathcal{R}$ as a definition over $\mathcal{S}$. We call these definitions *equivalent* and use $\equiv$ to show their equivalency. We call function δ_τ a *definition mapping* for τ. For instance, let $\mathcal{R}$ be the original schema and $\mathcal{S}$ be the alternative schema in Table 1, where $\tau : \mathcal{R} \rightarrow \mathcal{S}$. A learning algorithm may learn the following definition over $\mathcal{R}$: $h_\mathcal{R} = advancedStudent(X) \leftarrow inPhase(X, \text{“}post\_prelims\text{”}), yearsInProgram(X, \text{“}year\_5\text{”})$. If τ is definition preserving, then there must exist a function δ_τ such that $\delta_\tau(h_\mathcal{R}) = advancedStudent(X) \leftarrow student(X, \text{“}post\_prelims\text{”}, \text{“}year\_5\text{”})$. Transformation τ is *definition bijective* iff both τ and τ^{-1} are definition preserving. Clearly, in this case $\delta_{\tau^{-1}}$ and δ_τ^{-1} are equal.

Definition 3.1 *Algorithm A is schema independent under bijective and definition bijective transformation $\tau : \mathcal{R} \rightarrow \mathcal{S}$ iff for all $I_\mathcal{R}$ and $I_\mathcal{S}$ and training data E:*

- *δ_τ is bijective over $\mathcal{L}^A_\mathcal{R}$ and $\mathcal{L}^A_\mathcal{S}$.*
- *$A(\tau(I_\mathcal{R}), E) \equiv \delta_\tau(A(I_\mathcal{R}, E))$.*

The first condition in Definition 3.1 guarantees that algorithm A deals with equivalent hypothesis spaces over schemas $\mathcal{R}$ and $\mathcal{S}$. Algorithm A is schema independent under the set of transformations iff it is schema independent under all its members.

4 Schema Independence of Relational Learning Algorithms

A widely used family of transformations is vertical composition/decomposition (*composition/decomposition* for short) [1], which moves attributes around the relations of a schema. The transformation between the schemas in Table 1 is an example of vertical composition/decomposition. Given some general and frequently appearing FD and IND constraints in schemas $\mathcal{R}$ and $\mathcal{S}$, vertical composition/decomposition between $\mathcal{R}$ and $\mathcal{S}$ is both bijective and definition bijective [10]. We explore the schema independence of current algorithms over this set of transformations in this section.

Top-down algorithms search the hypothesis space from general to specific. The hypothesis space in top-down algorithms can be seen as a rooted directed acyclic graph in which nodes represent clauses and each arc is the application of a refinement operator. We analyze the schema independence properties of FOIL [11], an efficient and popular top-down algorithm that follows a greedy best-first search strategy. However, the results that we show hold for all top-down algorithms no matter which search strategy they follow. The graph for most schemas may grow significantly [11, 8]. Hence, the construction and search over the graph may become too inefficient to be practical. To be used in practice, algorithms restrict their search space, i.e. hypothesis space. A common method is to restrict the maximum length of each clause in the graph [11, 8]. Intuitively, because composition/decompositions modify the number of relations in a schema, equivalent clauses over the original and transformed schemas may have different lengths. Hence, this type of restrictions may result in different hypothesis spaces. One may like to fix this problem by choosing different values for the maximum lengths over the original and transformed schemas. The following theorem proves that it is not possible to achieve equivalent hypothesis spaces over the original and transformed schemas by restricting the maximum length of clauses *no matter what values are used over the original and transformed schemas*. Let $\tau : \mathcal{R} \rightarrow \mathcal{S}$ be a vertical composition/decomposition.

Theorem 4.1 *There is no definition mapping δ_τ for τ such that δ_τ is bijective over $\mathcal{L}_{\mathcal{R}}^{FOIL}$ and $\mathcal{L}_{\mathcal{S}}^{FOIL}$.*

One may modify FOIL so that, under some conditions, there exists a bijective definition mapping between hypothesis spaces over different schemas [10]. However, the modified of FOIL version has to evaluate clauses with rather large number of relations over all the training data. Since most of these clauses are already minimal, the algorithm may need to join large number of relations to evaluate each candidate clause. Hence, the learning may be very slow and not practical over relatively large databases. Further, FOIL traverses the graph, evaluates a set candidate clauses in the graph, and returns the most promising clause. Hence, to be schema independent, the algorithm must evaluate clauses at the same order over equivalent schemas. One of the operations of (modified) FOIL is assigning variables to attributes in the newly added relations in the current clause. It is not clear how FOIL can assign variables to attributes such that it maintains the same order of clauses over equivalent schemas without strong assumptions about the schemas, such as strong universal relation and unique attribute role assumptions [1]. Hence, the generate and test approach used in top-down algorithms like FOIL is generally at odds with schema independence.

Bottom-up algorithms search the hypothesis space from specific to general. In [10], we prove that current bottom-up algorithms are not schema independent. In this paper, we describe an extension to the bottom-up algorithm ProGolem [9], which is able to achieve schema independence. Given a positive example, bottom-up algorithms attempt to find the most specific clause in the hypothesis space that covers the example, relative to the database instance. This hypothesis is called the *bottom clause*. Then, they apply generalization operators on one or more of these bottom clauses. Let $\perp_{e,I}$ be the bottom clause associated with example e, relative to the database instance I. An algorithm for computing bottom clauses using inverse entailment is given in [8]. This algorithm takes as input the parameter $depth$, which specifies the maximum depth of any term in the bottom clause. Unfortunately, using the depth parameter does not result in equivalent bottom clauses associated with the same example, relative to equivalent instances of schemas that represent the same information. This is because different schemas require different depth values for equivalent clauses.

We propose the following modifications to the algorithm for generating bottom clauses. Given an input example, the algorithm creates the head of the clause by adding a literal with the same predicate name as the example, and with constants replaced by variables. A hash table is kept to map constants to variables. In each iteration, the algorithm looks for ground atoms in the database that contain constants stored in the hash table. For each ground atom, the algorithm creates a new literal with the same predicate name as the atom, and with (some) constants replaced by either variables specified by the hash table or new variables. If the newly created literal is not already in the clause, it is added. Then, the algorithm applies a modified version of the Chase algorithm [1].

Assume that the algorithm is generating the bottom clause relative to I. Assume that the algorithm selects relation R_i and adds a literal L_i to the bottom clause based on some tuple t_i of relation R_i. Let $\mathbf{L}$ be an inclusion class of R_i. For each constraint $R_i[\bar{A}_i] = R_k[\bar{A}_k]$ between the members of $\mathbf{L}$, the algorithm checks all tuples of relation R_k that share join attributes with t_i. For each tuple, the algorithm creates a literal L_k and assigns variables in the same way as it is done in the original algorithm. If there is no existing literal in the clause that has the same relation name and same old variables as L_k (only differs in new variables), then L_k is added to the clause. The algorithm ensures that the corresponding attributes in $\bar{A}_i$ and $\bar{A}_k$ are assigned the same variables. Because this version of Chase algorithm is terminal and enforces the available INDs with equality to the clause, the resulting clause is equivalent to the input clause [1].

We also propose a modification of the algorithm so that the stopping condition is based on a parameter called $maxvars$. This parameter indicates the maximum number of (distinct) variables in a bottom clause before starting a new iteration of the algorithm. At the end of each iteration, the algorithm checks the number of distinct variables contained in the bottom clause. If this number is less than the parameter $maxvars$, then the algorithm continues to the next iteration. Otherwise, the algorithm stops. Let $\tau : \mathcal{R} \rightarrow \mathcal{S}$ be a composition/decomposition. Let I and J be instances of $\mathcal{R}$ and $\mathcal{S}$, respectively, such that $\tau(I) = J$.

Theorem 4.2 *Let $\perp_{e,I}$ and $\perp_{e,J}$ be bottom clauses associated with e relative to I and J, respectively, generated by the algorithm described above. Then, $\perp_{e,I} \equiv \perp_{e,J}$.*

ProGolem is based on the *asymmetric minimal general generalization* ($armg$) operator, which is a generalization operator. ProGolem considers bottom clauses as ordered clauses. Therefore, to ensure

that the algorithm is schema independent, we must force clauses to have an equivalent order. One may use the content of the database instance to establish an order between inclusion classes, which is preserved under composition/decomposition. Let us define the natural join over an inclusion class in schema $\mathcal{R}$ as the join of all relations in $\mathcal{R}$ using their attributes that appear in INDs with equality. According to [10], τ does not join the relations from different inclusion classes in $\mathcal{R}$. Hence, one may use the natural joins over inclusion classes to define an order between inclusion classes in a database, which is preserved over all composition/decomposition transformations of the database. In this paper, we assume that equivalent bottom clauses have an equivalent order.

Theorem 4.3 *The $armg$ operator is schema independent under composition/ decomposition.*

We showed that we are able to get equivalent bottom clauses associated with the same example, relative to equivalent instances of schemas that represent the same information. We also showed that the $armg$ operator is schema independent. We denote by *ProGolem+Chase* the extended version of ProGolem that employs the modified bottom clause construction algorithm described above.

Corollary 4.4 *ProGolem+Chase is schema independent under composition/ decomposition.*

Query-based algorithms learn exact definitions by asking queries to an oracle , e.g., whether a data item belongs to the target concept [7, 2]. Because query-based algorithms follow a different learning model, Definition 3.1 is not suited for evaluating their schema (in)dependence. Query-based algorithms are theoretically evaluated by their *query complexity* – the asymptotic number of queries asked by the algorithm [7]. Therefore, we analyze the impact of schema transformations on the query complexity of these algorithms. We show that that a popular query-based algorithm called $A2$ [2] has drastically different asymptotic behavior over composition/decomposition. Specifically, we prove that the lower bound on the query complexity of $A2$ under one schema is greater than the upper bound on its query complexity under another schema.

Theorem 4.5 *Let $\Omega(f)_{\mathcal{R}}$ and $O(g)_{\mathcal{R}}$ be the lower bound and upper bound, respectively, on the query complexity of $A2$ for all target relations under schema $\mathcal{R}$, where f and g are functions of properties of $\mathcal{R}$. Then, there is a composition/ decomposition of $\mathcal{R}$, $\mathcal{S}$, such that $\Omega(f)_{\mathcal{R}} > O(g)_{\mathcal{S}}$.*

5 Empirical Results

Table 2 shows the results of learning the definitions for the $advisedBy(stud, prof)$ and $femaleActor(actor)$ relations over the UW-CSE and IMDb databases, respectively, using well-known algorithms. Our results indicate that these algorithms generally return different definitions with different degrees of effectiveness over different schemas of the same data. Although ProGolem seems to be schema independent over the UW-CSE database, its results vary with the schema over IMDb. ProGolem+Chase returns the same results over all schemas of both data sets and is generally as effective as ProGolem.

		UW-CSE				**IMDb**	
Algorithm	Metric	Original	4NF	Denorm. 1	Denorm. 2	Original	Single Lookup
FOIL	Precision	0.91	0.40	0.41	0.61	0.63	0.60
	Recall	0.73	0.44	0.55	0.92	0.43	0.44
ProGolem	Precision	0.86	0.86	0.86	0.86	0.89	0.98
	Recall	0.92	0.92	0.92	0.92	0.34	0.30
ProGolem+Chase	Precision	0.86				0.98	
	Recall	0.92				0.30	

Table 2: Results of learning relations over the UW-CSE (left) and IMDb (right) databases.

6 Conclusion

We defined the property of schema independence for relational learning and proved that current relational learning algorithms are not schema independent. We used the schema constraints to extend an existing algorithm to be schema independent under a widely used variation in schema design. We believe that this paper initiates some exciting theoretical investigations on the impact of database representation on the quality of learning concepts from the database.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1994.

[2] M. Arias, R. Khardon, and J. Maloberti. Learning Horn expressions with LOGAN-H. *J. Mach. Learn. Res.*, 8:549–587, Dec. 2007.

[3] R. Fagin. Normal Forms and Relational Database Operators. In *SIGMOD*, 1979.

[4] W. Fan and P. Bohannon. Information Preserving XML Schema Embedding. *TODS*, 33(1), 2008.

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

[6] R. Hull. Relative Information Capacity of Simple Relational Database Schemata. *SICOMP*, 15(3), 1986.

[7] R. Khardon. Learning function-free Horn expressions. *Machine Learning*, 37(3):241–275, 1999.

[8] S. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13:245–286, 1995.

[9] S. Muggleton, J. C. A. Santos, and A. Tamaddoni-Nezhad. ProGolem: A System Based on Relative Minimal Generalisation. In *ILP*, volume 5989, 2009.

[10] J. Picado, A. Termehchy, and A. Fern. Schema independent relational learning. *Technical Report*, arXiv:1508.03846, 2015.

[11] J. R. Quinlan. Learning Logical Definitions From Relations. *Machine Learning*, 5, 1990.

[12] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2002.

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