Schema Independence of Learning Algorithms

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

ABSTRACT

Learning novel concepts and relations from structured data sets, such as relational databases, is an important problem with many applications in data management and machine learning. It is well established that the same data set may be represented under different schemas due to various reasons, such as efficiency, data quality, and usability. Further, the schema of a database may evolve over time. In this paper, we argue that relational learning algorithms should be *schema independent*, i.e. they should return basically the same results across various schemas of the same data set. Schema independent relational learning algorithms require less manual tuning and are easier to use over real-world data sets. We formally define and explore this property for different types of relational learning algorithms. We analyze the schema independence of some popular relational learning algorithms both theoretically and empirically. Our results indicate that these algorithms are not generally schema independent and will deliver different results and accuracies or require different amount of training data over different schemas for the same data set.

1. INTRODUCTION

Learning novel concepts and relations from structured data sets, such as relational databases, is an important learning problem with a wide range of applications [23, 12, 20]. Given a relational database and training data, (statistical) relational learning algorithms learn the definitions for target concepts or relations as (weighted) first order logic formulas. Since it normally takes a prohibitively long time to explore all possible definitions for a target concept even for databases with relatively small number of relations, these algorithms selectively examine a "promising" subset of candidate definitions [15, 19, 22, 11]. The learning algorithms find the promising definitions according to their syntactic properties: they start with an initial set of clauses and iteratively generalize or specialize them by modifying their syntaxes. For example, they may replace constants (i.e. objects) or variables with some other variables in adding new relations to a clause to create new definitions.

Nevertheless, it has been well established that given some conditions that frequently occur in real world databases, people can and do represent the same data set using different schemas with diverse syntactic characteristics. [1, 21]. Table 2 shows two different schemas for the Mutagenesis dataset, which contains information about atoms, molecules, and their relationships and is widely used in inductive logic programming and statistical relational learning research [3, 18]. Relational learning researchers usually use the *Original schema* in Table 2 to represent the Mutagenesis dataset. This schema is (almost) in sixth normal form [4], which is generally discouraged in database literature due to its extremely poor performance for processing queries [1, 21]. It also contains a redundant attribute, *drug*, in relation *atm*. Hence, a professional database designer may choose *Schema 1* in Table 2 to represent this dataset.

Similarly, Table 1 shows some relations of the original and some alternative schemas for UW-CSE[1] database, which stores information about students and professors in a computer science department. This data set is widely used in statistical relational learning community to evaluate the effectiveness of statistical relational learning algorithms [11, 22]. The original schema of UW-CSE is in sixth normal form. Because each student, *stud*, has only one *phase* and *years* in relations *inPhase* and *yearsInProgram*, a professional database designer may compose relations *student*, *inPhase*, *yearsInProgram*, a new relation called *studentInfo*. She may also combine relations *professor* and *hasPosition* represent the database using *Schema 1* to speed up the query processing. In order to improve data quality and assure that each person involved in a project is either a student or a professor, she may also partition relation *project* to two relations *projectStud* and *projectProf* and define foreign key constraints between attribute *stud* in relations *student* and *projectStud* and attribute *prof* in relations *professor* and *projectProf* and store the data set in form of *Schema 2*. If the data becomes rather large, she may further refine *Schema 2* to *Schema 3* or *Schema 4* to improve the performance of answering queries over the database.

People may choose to represent their data in one way or another for several reasons. It is easier to enforce some integrity constraints over decomposed schemas such as the original schema or *Schema 1* for UW-CSE data set [1]. However, because these schemas contain too many relations, they are hard to understand and maintain. It also takes a long time to answer queries over databases with such schemas [1, 21]. Thus, one may sacrifice data quality and choose more *composed* (or *denormalized*) schemas, such as *Schema 3* or *Schema 4* for UW-CSE data set to achieve better usability and/or performance. She may partition each relation to its subsets (i.e. horizontal decomposition) to distribute the data and enforce data integrity constraints or union relations to create a more usable schema with smaller number of relations. A database designer may also hit a middle ground by choosing a style of representation for some re-

[1] http://alchemy.cs.washington.edu/data/uw-cse

Original Schema	Schema 1	Schema 2	Schema 3	Schema 4
student(stud) inPhase(stud,phase) yearsInProgram(stud,years)	student(stud) studInfo(stud,phase,years)	student(stud,phase,years)	student(stud,phase,years)	student(stud,phase,years,prj)
project(prj,person)	project(prj,person)	projectStud(prj,stud)	project(prj,stud,prof)	
professor(prof) hasPosition(prof,position)	professor(prof,position)	professor(prof,position) projectProf(prj,prof)	professor(prof,position)	professor(prof,position,prj)

Table 1: The original and alternative schemas of the UW-CSE data set.

Original Schema	Schema 1
bond(drug,atom1,atom2,btype) atm(drug,atom,elem,atmtype,charge) logp(drug,hydrophob) lumo(drug,energy)	bond(drug,atom1,atom2,btype) atm(atom,elem,atmtype,charge) comp(drug,energy,hydrophob)

Table 2: Different schemas for the Mutagenesis dataset.

lations and another style for other relations in the schema. Further, as the relative priorities of these objectives changes over time, the schema will also evolve.

Since relational learning algorithms search for the definition of target relations by modifying syntactic properties of clauses, they may learn different definitions over different schemas for the same data set. For instance, many top-down relational learning algorithm such as FOIL start their search by creating a clause with one relation and try different variables as its arguments [19, 23]. They pick the promising clauses that may lead to the desired definition using their performances over training data and repeats the process by adding new relations to these clauses. Thus, it may generate and examine different sets of clauses over various schemas for the same database, and deliver a different result over each schema. According to our experiments, FOIL defines relation *advisedby(stud, prof)* based on co-authorship between student and professor over a normalized schema such as *Schema 2*, for UW-CSE data set. Nonetheless, it defines the same target relation based on the courses a student takes with the professor over a denormalized schema like *Schema 4*. As heterogeneity in representation is a widely accepted property of real-world data sets [27], current relational learning algorithms may not scale to work over large number of data sets without experts' interventions.

We argue that a relational learning algorithm should be *schema independent*, i.e. it should deliver essentially the same definition for the target concept or relation across various schemas for the background knowledge (i.e. database) given the same training data. A schema independent learning algorithm is guaranteed to be effective over the data sets that are organized differently from the ones used to test it. It requires less manual tuning to apply and maintain over new data sets, therefore, it can be used over large number of data sets. It can also use background knowledge and training data from multiple data sources with different styles of representation more easily an effectively.

To the best of our knowledge, the property of schema independence has not been explored for relational learning algorithms. Our contributions in this paper are as follows:

- We introduce and formally define the property of schema independence and explore its benefits for relational learning algorithms. We define the degree of schema independence for a relational learning algorithm as its robustness over transformations that modify the database schema and preserve its information content.
- We theoretically analyze the degree of schema independence for some of the well known relational learning algorithms over a frequently used family of schema transformations called *vertical decomposition*. Our investigation suggests that these transformations considerably modify the maximum amount of training data required by these algorithms.
- We also empirically study the schema independence of some relational learning algorithms under vertical and horizontal decomposition transformations. Our results indicate that transforming schema considerably affects the effectiveness, efficiency, and query complexities for these algorithms.

This paper is organized as follows. Section 2 describes the basic definitions and related works. Section 3 introduces and formally defines the property of schema independence for relational learning algorithms. Section 4 analyzes the impact of some transformations on the maximum amount of training data required to learn concepts by some learning algorithms. Section 5 contains our empirical results and Section 6 concludes the paper.

2. BACKGROUND

2.1 Related Work

Researchers in database systems have explored the relationship between query answering algorithms and different representations for a database [5, 14, 28]. They have argued in favor of the query answering algorithms whose results do not depend on specific representations of the database, as they can be scaled and used over a large number of databases without experts' interventions. Whether this property can be achieved for relational learning algorithms has not been explored. Our paper explores the relationship between data representation and accuracy and sample complexity of learning algorithms to answer these questions for relational learning algorithms. Schema independence extends the notion of logical data independence [5] for relational learning algorithms.

In order to achieve logical data independence for database queries, some researchers have proposed to convert schemas to a common format, namely universal relation [14]. Queries over original schemas need to be also translated to queries over the common format. This approach considers only the variations in data representation created by assigning attributes to different relations. Nevertheless, the heterogeneity in organizing data may be caused by other types of schema modifications [1]. It is not also clear if every schema can be represented in such a common format [1, 13]. Transforming various schemas and queries to a common format may itself require some experts' interventions, which is not desirable and reduces the degree of logical data independence. Hence, we argue for representation independence of the learning algorithms instead of finding a common format and transforming all schemas to such form.

One may define new predicates based on schema relations and use them to facilitate relational learning [24, 26]. We, however, explore the effects of schema representations on learning.

2.2 Basic Definitions

Let $Attr$ be a countably infinite set of symbols that contains the names of *attributes*. The *domain* of attribute A is a countably infinite set of values (i.e. constants or objects) that A may contain. We assume that all attributes share a single domain dom. A *relation* is a finite subset of $Attr$. We use terms predicate and relation interchangeably. A *tuple* over relation R is a total map from the set of attributes in R to dom. The relation instance I_R of relation R is a finite set of tuples. A *constraint* restricts the properties of data stored in a database. Examples of constraints are *functional dependencies* and *inclusion dependencies*. Functional dependency $A \rightarrow B$ 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. Inclusion dependency between attribute $C \in R$ and $D \in S$, denoted as $C \subset D$, states that in all instances of I_R and I_S values of attribute C must also appear attribute D. A *schema* is a pair $\mathcal{R} = (\mathbf{R}, \Sigma)$, where $\mathbf{R}$ is a finite set of relations and Σ is a finite set of constraints. An instance of schema $\mathcal{R}$ is a mapping I over $\mathcal{R}$ that associates each relation $R \in \mathcal{R}$ to a relation instance I_R.

A *literal* is a relation, or the negation of a relation. A *definite Horn clause* (Horn clause or clause, for short) is a finite set of literals that contains exactly one positive literal. The positive literal is called the head of the clause, and the set of negative literals is called the body. The head, which does not belong to schema $\mathcal{R}$, is the *target relation*. The literals in the body are relations that belong to schema $\mathcal{R}$. A clause has the form:

$$S(u) \leftarrow L_1(u_1), \cdots, L_n(u_n).$$

where $n > 0$, L_j, $1 \leq j \leq n$, is $R_j \in \mathcal{R}$, and u and u_j are sets of variables or constants. A *Horn expression* is a set of Horn clauses. A *Horn definition* is a Horn expression with the same predicate in the heads of all clauses.

3. FRAMEWORK

We adapt the notion of equivalency between schemas from [9, 7] to find if two schemas represent the same information. We denote the set of instances of schema $\mathcal{S}$ as $\mathcal{I}(\mathcal{S})$. Given schemas $\mathcal{S}$ and $\mathcal{R}$, a *transformation* is a mapping $\tau : \mathcal{I}(\mathcal{R}) \rightarrow \mathcal{I}(\mathcal{S})$. For brevity, we write transformation τ as $\tau : \mathcal{R} \rightarrow \mathcal{S}$. Schema $\mathcal{S}$ *dominates* schema $\mathcal{R}$ via transformation $\tau : \mathcal{R} \rightarrow \mathcal{S}$, iff transformation $\tau^{-1} : \mathcal{S} \rightarrow \mathcal{R}$ exists and the composition of τ and τ^{-1} is the identity mapping on $\mathcal{I}(\mathcal{R})$. In this case, we call transformation τ *invertible*. Schemas $\mathcal{S}$ and $\mathcal{R}$ are *equivalent* if and only if they dominate each other. If $\mathcal{S}$ and $\mathcal{R}$ are equivalent, one can convert instance $I \in \mathcal{I}(\mathcal{S})$ to instance $J \in \mathcal{I}(\mathcal{R})$ and reconstruct I from the available information in J. Thus, equivalency between two schemas indicates that they essentially represent the same information.

EXAMPLE 3.1. *Let the following functional and inclusion dependencies hold over the relations of Original Schema that are shown in Table 1:* $inPhase.stud \subseteq yearsInProgram.stud$, $yearsInProgram.stud \subseteq inPhase.stud$, $stud \rightarrow phase$, $stud \rightarrow years$. *One may join relations* $inPhase$ *and* $yearsInPrograms$ *to map each instance of the Original Schema to an instance of Schema 1. Further, each instance of Schema 1 can be mapped to an instance of the Original Schema by projecting relation* $studInfo$ *to relations* $inPhase$ *and* $yearsInProgram$*(i.e. vertical decomposition). Hence, these schemas are equivalent.*

We would like to learn essentially same Horn expressions for target relations over schemas that represent the same information. A relational learning algorithm generally may limit its hypothesis space by choosing certain hypothesis language for its target relations' definitions. For instance, it may consider only clauses whose numbers of literals are fewer than a given number. In order to learn the same definition for a target relation, one should be able to express essentially same definitions for a target relation over equivalent schemas. More formally, let $\mathcal{L}$ be the hypothesis language that is a subset of Horn expressions. Invertible transformation $\tau : \mathcal{R} \rightarrow \mathcal{S}$ is *definition preserving* w.r.t. $\mathcal{L}$ iff there exists a function $\delta : \mathcal{L} \rightarrow \mathcal{L}$ such that for every definition $p \in \mathcal{L}$ and $I \in \mathcal{I}(\mathcal{R})$ $p(I) = \delta(p)(\tau(I))$. Schemas $\mathcal{S}$ and $\mathcal{R}$ are *definition equivalent* iff there are definition preserving transformations from $\mathcal{S}$ to $\mathcal{R}$ and from $\mathcal{S}$ to $\mathcal{R}$.

Composition of two Horn expressions p and q, denoted as $p \circ q$, is a Horn expression created by applying p to the head predicates of q [1]. Language $\mathcal{L}$ is closed under composition with Horn expression q iff the composition of every definition $p \in \mathcal{L}$ and q, $p \circ q$, belongs to $\mathcal{L}$.

PROPOSITION 3.2. *Given schemas $\mathcal{S}$ and $\mathcal{R}$, invertible transformation $\tau : \mathcal{S} \rightarrow \mathcal{R}$, and language $\mathcal{L}$, if $\mathcal{L}$ is closed under composition with τ^{-1}, for each definition $p \in \mathcal{L}$ over $\mathcal{S}$ there is a definition $r \in \mathcal{L}$ over $\mathcal{R}$ such that $r = p \circ \tau^{-1}$.*

Proof: The proof is similar to the proof of Theorem 2.1 in [7].

According to Proposition 3.2, if schemas $\mathcal{R}$ and $\mathcal{S}$ are equivalent and $\mathcal{L}$ is closed under the composition with the invertible transformations from $\mathcal{R}$ to $\mathcal{S}$ and $\mathcal{S}$ to $\mathcal{R}$, each definition over $\mathcal{S}$ can be rewritten as a definition over schema $\mathcal{R}$ and vice versa. We call these definitions *equivalent*.

EXAMPLE 3.3. *Let the hypothesis language be the set of Horn definitions. Because this language is closed under composition with join and projection, the Original Schema and Schema 1 in Example 3.1 are definition equivalent.*

Sample based relational learning algorithms take labeled examples and background knowledge (i.e. database instance), and learn first-order clausal theories. This type of relational learning algorithms are normally studied in the context of Inductive Logic Programming (ILP) and statistical relational learning (SRL). In this paper, we focus on the algorithms whose hypothesis language is a subset of or equal to Horn expressions. We define the property of schema independence for sample based learning algorithms.

DEFINITION 3.4. *A sample based learning algorithm is schema independent iff it learns equivalent definitions for the same target relation over equivalent schemas given the same training data.*

Several algorithms have been proposed to perform exact learning via queries in relational domains [10, 25]. These algorithms differ from the standard supervised learning task in that they learn by asking queries to an oracle, instead of taking a set of labeled examples as input. The questions are usually of types equivalence queries (EQ) and membership queries (MQ). We consider the algorithms whose hypothesis language is a subset of or equal to Horn expressions. We define the property of schema independence for query based learning algorithms as follows.

DEFINITION 3.5. *A query based learning algorithm is schema independent iff it learns equivalent definitions for the same target relation over equivalent schemas by asking the same numbers of EQs and MQs.*

One may propose a stronger condition for schema independence that requires query based learning algorithms to ask similar queries to learn the same relation across equivalent schemas. However, in this paper we consider the main source of variation to be the number of queries asked to the oracle.

Algorithm	EQs	MQs
A2 [10]	$2mpk^{a+k}$	$m^2pk^{a+3k} + nmpk^{a+k}$
Learn-MQ [25]	pmk^a	$pmk^a(n + mk^k)$

Table 3: Query complexities of some query-based relational learning algorithms.

4. SCHEMA INDEPENDENCE OF QUERY-BASED ALGORITHMS

Khardon [10] studied the task of learning function-free first-order Horn expressions using the query based learning model and proposed a query based algorithm called A2. A2 uses a pairing operation similar to least general generalization (lgg) to construct clauses, considering only pairs of literals which guarantee an injective mapping between variables. Following this algorithm, Selman and Fern [25] developed an algorithm called Learn-MQ in the learning from entailment setting that performs exact learning of first-order definite theories via queries. The number of queries asked by these algorithms to learn a relation over a database is provably bounded by properties of the database schema [10, 25]. Table 3 shows the maximum number of queries asked by these algorithms. The parameters p and a denote the number of relations in the schema and the largest arity of any relation in the schema, respectively. The parameter k denotes the largest number of variables in a clause in the definition of the target relation. The number of clauses in the definition of the target relation is shown by m. The largest number of constants (i.e. objects) in any example is denoted by n.

In this section we analyze the sensitivity of the upper bounds on the number of queries asked by these algorithms, i.e. their query complexities, to a well known family of database transformation called (lossless) vertical decomposition [1]. These transformations are among the most frequently used transformations over relational databases. We show a case where there is a considerable difference in the number of queries required under two equivalent schemas. Consider schema $\mathcal{R} = (\mathbf{R}, \Sigma)$, where for each relation $R(A_1, \cdots, A_l) \in \mathbf{R}$, we have $l > 2$ and there are $l - 1$ functional dependencies $A_1 \rightarrow A_i$, $2 \leq i \leq l$, in Σ. Let $\mathcal{S} = (\mathbf{S}, \Omega)$ be another schema, such that for every relation $R(A_1, \cdots, A_l) \in \mathbf{R}$, we have $l - 1$ relations in $\mathbf{S}$ in form of $S_i(A_1, A_i)$, $2 \leq i \leq l$. For each relation $S_i(A_1, A_i) \in \mathbf{S}$, Ω contains the functional dependency $A_1 \rightarrow A_i$. For each set of relations $S_i(A_1, A_i)$, $2 \leq i \leq l$, Ω also contains $2(l-1)$ inclusion dependencies in form of $S_2.A_1 \subseteq S_j.A_1$ and $S_j.A_1 \subseteq S_2.A_1$, $2 < j \leq l$. It is straightforward to show that $\mathcal{R}$ and $\mathcal{S}$ are equivalent [1]. The transformation that maps the instances of $\mathcal{R}$ to $\mathcal{S}$ is projection (i.e. decomposition) and the one that maps $\mathcal{S}$ to $\mathcal{R}$ is join. The hypothesis languages of both A2 and Learn-MQ are closed under composition with these transformations.

Let $p(\mathcal{R})$ be the number of relations in $\mathcal{R}$, $a(\mathcal{R})$ be the largest arity of any relation in $\mathcal{R}$, and $k(\mathcal{R})$ be the largest number of variables in a rule that defines the target relation over $\mathcal{R}$. We define $p(\mathcal{S})$, $a(\mathcal{S})$, and $k(\mathcal{S})$ analogously. Let the target relation t be a relation in $\mathcal{S}$. If the number of relations in $\mathcal{R}$ is $p = p(\mathcal{R})$ and the maximum arity is $a = a(\mathcal{R})$, then the maximum number of relations in $\mathcal{S}$ is $p(\mathcal{S}) = p(a-1)$ as each relation is decomposed to at most $a - 1$ relations. We also have $a(\mathcal{S}) = 2$. Since the target relation is a relation in $\mathcal{S}$, the largest number of variables in a rule for its definition over $\mathcal{S}$, $k(\mathcal{S}) = a(\mathcal{S}) = 2$. This value over $\mathcal{R}$ is $k(\mathcal{R}) = a(\mathcal{R}) = a$. The number of clauses in the definition of target relation, m, is equal to 1 for both schemas. Each relation over $\mathcal{S}$ has smaller arity than its mapped relation in $\mathcal{R}$. Thus, the value of n over $\mathcal{R}$, $n(\mathcal{R})$, is greater than or equal to the value of n over $\mathcal{S}$, $n(\mathcal{S})$.

Under schema $\mathcal{R}$, the A2 algorithm would require at most $2mpa^{2a}$ EQs and $m^2pa^{4a} + n(\mathcal{R})mpa^{2a}$ MQs. On the other hand, under schema $\mathcal{S}$, this algorithm requires at most $2mp(a-1)2^4$ EQs and $m^2p(a-1)2^8 + n(\mathcal{S})mp(a-1)2^4$ MQs. Under schema $\mathcal{R}$, the Learn-MQ algorithm would require at most pma^a EQs and $pma^a(n(\mathcal{R}) + ma^a)$ MQs. On the other hand, under schema $\mathcal{S}$, this algorithm require at most $p(a-1)m2^2$ EQs and $p(a-1)m2^2(n(\mathcal{S}) + m2^2)$ MQs. The number of EQs and MQs for schema $\mathcal{R}$ grows exponentially with the largest arity, while the number of EQs and MQs for schema $\mathcal{S}$ grows linearly with this property. Therefore, if we choose the value of a sufficiently large, we observe considerable differences between the worst case query complexities of both algorithms over schema $\mathcal{R}$ and $\mathcal{S}$.

5. EMPIRICAL RESULTS

5.1 Sample-based systems

5.1.1 Experimental settings

In this section we evaluate the impact of representation on sample-based relational learning algorithms using Aleph[2]. Aleph is a well known ILP system that can emulate several other ILP systems. In this paper, we use it to partially emulate FOIL [19] and Progol [15]. We carried experiments on the UW-CSE data set by Richardson and Domingos [22], which consists of 12 relations, 2673 tuples, and 113 positive examples. Following [8], we generated negative examples using the closed-world assumption, and then sampled these to obtain twice as many negative examples as positive examples. Following the original authors of the data set, we divided the data into 5 folds, each one corresponding to a different group of the CSE department, and we tried to learn the relation *advisedBy(stud,prof)*, which indicates that student *stud* is advised by professor *prof*.

We have represented the data set using three equivalent schemas that are mapped to each other via vertical decomposition (i.e. projection) and composition (i.e. join) transformations. We employed the original schema, schema 2 and schema 4 shown in Table 1. For schema 2, we replaced relations *inPhase* and *yearsInProgram* with the composed relation *studentInfo*, we removed relations *professor* and *student*, and replaced the relations *project* and *publication* with *projectStud*, *projectProf*, *publicationStud*, and *publicationProf* in the original schema, respectively. Schema 4 is the result of a vertical composition of schema 2. The original schema contains 10 relations, schema 2 contains 9 relations, and schema 4 contains 3 relations.

For these experiments, we used a configuration similar to the one used in [8, 6], with a few variations, as shown in Table 4. For the original schema and schema 2, we set the upper bound on layers of new variables (*i*) to 2, and for schema 4 we set it to 1. With these numbers, the search spaces using all schemas are equivalent. We varied the number of nodes to be explored when searching for an acceptable clause (*nodes*) to evaluate how the size of the search space affects the effectiveness over different schemas. We also varied the beam size to emulate FOIL (*openlist = 1*) and Progol (*openlist = infinite*). When we varied the beam size, we set the number of nodes to 100,000. For the rest of the parameters, we used their default values.

In these experiments, we evaluate accuracy, precision, recall, and running time, showing the average over 5-fold cross validation. We run Aleph on a high performance computing cluster, which con-

[2]http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html

Parameter	Value
Least positive coverage of a valid clause (*minpos*)	10
Maximum number of literals in a valid clause (*clause-length*)	5
Maximum number of negative examples covered by a valid clause (*noise*)	1000
Search strategy (*search*)	heuristic
Least positive coverage of a valid clause (*minpos*)	10
Evaluation function (*evalfn*)	compression
Layers of new variables (*i*)	1 and 2

Table 4: Aleph parameters.

tains a heterogeneous mix of dual processor servers, with a range between 4 and 48 GB of main memory, and running RedHat Enterprise Linux 6. Running times are shown in seconds (s) and minutes (m).

5.1.2 Results

Table 5 shows the results of varying the number of nodes in the search space and Table 6 shows the results of varying the beam size, both under three different, but equivalent schemas. We first show an example where, when emulating FOIL, Aleph learns different clauses over the original schema and schema 4. Aleph learns the following clauses over the original schema:

(1) $advisedBy(A, B) \leftarrow$
$inPhase(A, post\_quals), publication(C, B).$
(2) $advisedBy(A, B) \leftarrow$
$inPhase(A, post\_generals), publication(C, B), publication(C, A).$

However, under the same training data, Aleph learns the following clauses over schema 4:

(3) $advisedBy(A, B) \leftarrow$
$studentInfo(A, post\_quals, C, D, E).$
(4) $advisedBy(A, B) \leftarrow$
$course(C, B, A, D, E), course(F, B, G, H, level\_500).$

The learned clauses over both schemas show that the student must be in an advanced phase. This indicates that according to the training examples, mostly advanced students have an advisor. Over schema 4, a common relation appearing in the learned clauses is *course*, and it states that the student has been TA for a course offered by his/her advisor, as shown in clause (4). FOIL is not able to learn clauses that express this information under the original schema because it would require to first learn that a course C is taught by professor B, and then learn that student A was a TA for course C. Because FOIL is a greedy algorithm, it selects other clauses with higher score. On the other hand, a common concept appearing in clauses learned under the original schema is that of the advisor and student having a common publication, as shown in clause (2). However, FOIL is not able to learn this concept under schema 4 because because clauses that express different information, such as clause (4), get a better score.

As shown in the previous example, under schema 4 FOIL usually learns clauses that state that the student has been TA for a course offered by his/her advisor, which is not always the case. This makes FOIL have a low accuracy, precision and recall under schema 4. However, under the original schema and schema 2, FOIL gets similar scores. This shows that if small changes are made to the schema, and these changes don't affect how the clauses are learned (such as requiring chains), FOIL is not affected by the design. However if bigger changes are made to the representation, such as schema 4, FOIL performs differently, thus it is not representation independent.

When emulating Progol, Aleph also learns different clauses over different schemas. Consider the following clauses learned under the original schema:

(1) $advisedBy(A, B) \leftarrow$
$publication(C, B), publication(C, A).$
(2) $advisedBy(A, B) \leftarrow$
$inPhase(A, post\_quals), publication(C, B).$
(3) $advisedBy(A, B) \leftarrow$
$inPhase(A, post\_generals), taship(C, A, D), publication(E, B).$

Under the same training data, Aleph learns the following clauses over schema 4:

(4) $advisedBy(A, B) \leftarrow$
$professorInfo(B, faculty, C, D), studentInfo(A, E, F, G, D).$
(5) $advisedBy(A, B) \leftarrow$
$course(C, B, D, E, level\_500), course(F, B, G, H, level\_400),$
$studentInfo(A, post\_quals, I, J, K).$

Sometimes Progol is able to learn clauses that express the same information under different schemas. For example, clauses (1) and (4) express the concept of the student and the advisor having a common publication. However this is not always the case. Taking a look at the clauses learned under the original schema and schema 2, more general clauses are learned, usually expressing information about professors or students, but not always a relationship between them. This results in a high recall, but low precision. On the other hand, clauses learned under schema 4 are able to capture more information with less relations, as they have a bigger arity. Therefore, these clauses are more precise, but less general, resulting in a lower recall. Because the learned clauses are not equivalent under different schemas, according to our definition, we can conclude that Progol is not representation independent either.

With the parameters mentioned above, most of the running time is spent in performing search over the search space. Because FOIL is greedy, it runs very quickly over all schemas, even for a large number of nodes. On the other hand, Progol's running time is dependent on the size of the search space. The difference of running time over different schemas is mostly seen with a large number of nodes. Because we set schema 4 to have only one layer of variables, it is the fastest, as it spends less time creating the bottom clause and traversing the search space. Schema 2 takes more time because it requires two layers of variables and its relations have higher arity, which results in bigger bottom clauses.

5.2 Query-based systems

5.2.1 Experimental settings

In this section, we evaluate the impact of design on query-based systems. We employed the LogAn-H system [2], which is an implementation of the A2 algorithm [10]. Specifically, we employed the interactive algorithm with automatic user mode. In this mode, the system is told the expression to be learned, and the algorithm's queries are answered automatically using this expression.

We generated random expressions over schema 4 in Table 1, each containing multiple definite Horn clauses. The only parameter for generating each clause is the number of variables in the clause. All generated expressions are Horn definitions. To generate the head of each clause, we created a new relation of random arity, where the minimum arity is 1 and the maximum arity is the maximum arity of the relations in schema 4. The body of each clause can be of any length as long as the number of variables in the clause is equal to the specified parameter and all variables appearing in the head relation also appear in any relation in the body. The body of the clause is composed of randomly chosen relations, where each relation can be the head relation (allowing for recursive clauses) or any relation in the input schema. Head and body relations are populated with

Nodes	Accuracy			Precision			Recall			Running time		
	Original Schema	Schema 2	Schema 4	Original Schema	Schema 2	Schema 4	Original Schema	Schema 2	Schema 4	Original Schema	Schema 2	Schema 4
1000	0.73	0.76	0.77	0.62	0.62	0.71	0.72	0.81	0.60	5.42s	5.00s	3.10s
3000	0.74	0.77	0.77	0.62	0.62	0.71	0.72	0.81	0.60	12.83s	11.66s	9.09s
5000	0.73	0.77	0.77	0.62	0.62	0.71	0.72	0.81	0.60	20.19s	19.29s	16.55s
100000	0.76	0.75	0.78	0.62	0.60	0.73	0.76	0.76	0.60	3.76m	7.16m	2.41m

Table 5: Results for the original schema, schema 2 and schema 4, varying the number of nodes in the search space.

Beam size	Accuracy			Precision			Recall			Running time		
	Original Schema	Schema 2	Schema 4	Original Schema	Schema 2	Schema 4	Original Schema	Schema 2	Schema 4	Original Schema	Schema 2	Schema 4
1	0.72	0.72	0.64	0.62	0.64	0.48	0.70	0.71	0.43	2.34s	2.71s	0.75s
3	0.72	0.75	0.76	0.62	0.61	0.70	0.70	0.81	0.62	2.64s	2.12s	0.72s
infinite	0.76	0.75	0.78	0.62	0.60	0.73	0.76	0.76	0.60	3.76m	7.16m	2.41m

Table 6: Results for the original schema, schema 2 and schema 4, varying the beam size.

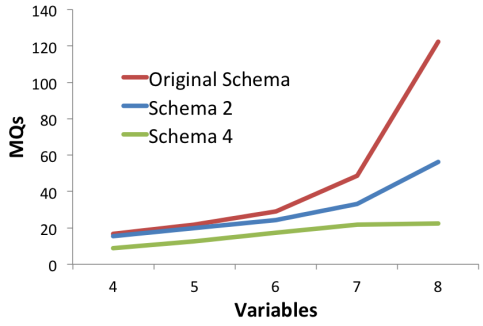

Figure 1: Average number of membership queries required by the A2 algorithm.

variables, where each variable is randomly chosen to be a new (until reaching the input number of variables) or already used variable. Clauses cannot contain function or constant symbols.

After generating each random expression over schema 4, we translate these expressions to the original schema and schema 2 by simply doing vertical decomposition to each of the clauses in an expression. Then, we minimized all expressions using the Homomorphism theorem and the Chase algorithm with the functional and inclusion dependency constraints [1].

We varied the number of clauses in an expression to be between 1 and 5, each containing between 4 and 8 variables. We generated 50 random expressions for each setting, getting a total of 250 expressions for each number of variables. The A2 algorithm takes as input the target expression and the signature. The signature consists of the names of all relations in the input schema and the head relation, as well as the arity of each relation. We ran the LogAn-H system with the original expression over schema 4 and the translated expressions over the original schema and schema 2, and recorded the number of queries required to learn each expression. In these experiments, we report the query complexity – number of equivalence queries (EQs) and membership queries (MQs) – of the A2 algorithm.

5.2.2 Results

The average number of EQs required by the A2 algorithm to learn Horn expressions over all schemas is constant over different numbers of variables. For all schemas, the number of EQs is almost the same, with a maximum difference of 2 queries. On the other hand, there are considerable differences in the number of MQs. Figure 1 shows the number of MQs required by the A2 algorithm over the three schemas. The difference in MQs between schemas comes from the step that performs the minimization of objects in the A2 algorithm. Given a negative example, the algorithm iterates over domain elements. In each iteration, it drops an element and all relations in which the element appears, and asks a MQ to check if the interpretation is negative. If it is, then the algorithm continues with the smaller example. In the original schema, relations have smaller arities. This allows dropping objects and relations without emptying the interpretation. This is not the case for schema 4, where dropping a relation may empty the interpretation, as relations have bigger arity. The algorithm does not ask MQs for an empty interpretation. Schema 2 requires more MQs than schema 4, but less than the original schema, as it is in the middle ground in terms of composition. Therefore, the algorithm asks more MQs when using the most decomposed schema, which is the original schema.

These results differ from the worst-case analysis shown in Section 4. Target expressions were generated for schema 4 and then translated to the original schema and schema 2. Therefore, these expressions are equivalent, contain the same number of variables, and only differ in the maximum arity of relations. According to the worst-case analysis, A2 should require more queries to learn the target expressions under schema 4. This is not the case in the average-case, as seen in the empirical results. This issue requires further investigation.

6. CONCLUSION

We introduced the concept of schema independence, and argued that relational learning algorithm should be schema independent. We explored the effects of different representations of data on query-based and sample-based learning algorithms. Our results showed that different representations considerably affect the effectiveness and efficiency of sample-based algorithms, as well as the query complexity of query-based algorithms. Worst-case analysis differed from average-case results for query-based analysis. Hence, understanding this difference would be helpful for studying the effect of different representations on these and other algorithms. It would be interesting to evaluate the impact of different representations on sample-based, bottom-up algorithms, such as Golem [16], ProGolem [17] and the batch algorithm of LogAn-H [2]. Finally, developing a schema independent algorithm is the ultimate goal of this research.

7. ACKNOWLEDGMENTS

We thank Roni Khardan for helping us with LogAn-H system. Jose Picado is supported by a Rickert Scholarship Award.

8. 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] Ashwin Srinovasan and Stephen Muggleton and M. Sternberg and R. King. Theories for Mutaginicity: A Study in First Order and Feature Based Indeuction. *Artificial Intelligence*, 1996.

[4] Chris Date and Darwen Hugh and Lorentzos Nikos. *Temporal Data and the Relational Model: A Detailed Investigation into the Application of Interval and Relation Theory to the Problem of Temporal Database Management*. Oxford: Elsevier LTD., 2003.

[5] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *CACM*, 13(6):377–387, 1970.

[6] J. Davis, E. S. Burnside, I. de Castro Dutra, D. Page, and V. S. Costa. An integrated approach to learning bayesian networks of rules. In *ECML*, volume 3720, pages 84–95, 2005.

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

[8] M. V. M. França, G. Zaverucha, and A. S. d'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning*, 94:81–104, 2014.

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

[10] R. Khardon. Learning function-free horn expressions. *Mach. Learn.*, 37(3):241–275, Dec. 1999.

[11] Lilyana Mihalkova and Raymond Mooney. Bottom-Up Learning of Markov Logic Network Structure. In *ICML*, 2007.

[12] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[13] Luc De Raedt. Attribute-Value Learning Versus Inductive Logic Programming: The Missing Links. In *ILP*, 1998.

[14] D. Maier et al. Toward Logical Data Independence: a Relational Query Language Without Relations. In *SIGMOD*, 1982.

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

[16] S. Muggleton and C. Feng. Efficient induction of logic programs. In *New Generation Computing*. Academic Press, 1990.

[17] S. Muggleton, J. C. A. Santos, and A. Tamaddoni-Nezhad. Progolem: A system based on relative minimal generalisation. In *ILP*, volume 5989, pages 131–148, 2009.

[18] Niels Landwehr and Kristian Kersting and Luc De Raedt. nFOIL:Integrating Naive Bayes and FOIL. In *AAAI*, 2005.

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

[20] L. D. Raedt. *Logical and Relational Learning*. Springer, 2008.

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

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

[23] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 2009.

[24] S. Kramer. Predicate invention: A comprehensive view, 1995.

[25] J. Selman and A. Fern. Learning first-order definite theories via object-based queries. ECML PKDD'11, pages 159–174, 2011.

[26] Stanley Kok and Pedro Domingos. Statistical Predicate Invention. In *ICML*, 2007.

[27] Surajit Chaudhuri. How Different is Big Data? In *ICDE*, 2012.

[28] A. Termehchy, M. Winslett, and Y. Chodpathumwan. How Schema Independent Are Schema Free Query Interfaces? In *ICDE*, 2011.