2018 IEEE 34th International Conference on Data Engineering

AutoMode: Relational Learning With Less Black Magic

Jose Picado, Sudhanshu Pathak, Arash Termehchy, Alan Fern
School of EECS, Oregon State University
{picadolj,pathaks,termehca,alan.fern}@oregonstate.edu

***Abstract*—Relational learning algorithms learn the Datalog definition of novel relations in terms of existing relations in the database. In order to effectively use these algorithms, users must constraint the space of candidate definitions by specifying a language bias. Unfortunately, specifying the language bias takes a great deal of time and effort, as it is done via trial and error and is guided by the expert's intuitions. We demonstrate *AutoMode*, a system that leverages information in the schema and content of the database to automatically induce the language bias used by popular relational learning algorithms.**

I. Introduction

Learning novel concepts or relations over relational databases has attracted a great deal of attention due to its applications in machine learning and data management [3]. Consider the UW-CSE database (*alchemy.cs.washington.edu/data/uw-cse*), which contains information about a computer science department and its schema fragments are shown in Table I. One may want to predict the new relation *advisedBy(stud,prof)*, which indicates that the student *stud* is advised by professor *prof*. Given the UW-CSE database and positive and negative training examples of the *advisedBy* relation, relational learning algorithms attempt to find a definition of this relation in terms of the existing relations in the database [3], [4]. Learned definitions are usually first-order logic formulas and often restricted to Datalog programs. For example, a relational learning algorithm may learn the following Datalog program for the *advisedBy* relation: $advisedBy(x, y) \leftarrow publication(z, x), publication(z, y)$, which indicates that a student is advised by a professor if they have been co-authors of a publication.

Relational learning algorithms can exploit the relational structure of the data, making them useful for domains where structure of data is important. Moreover, their learned definitions are interpretable and easy to understand. Relational learning has several applications in database management and machine learning, such as learning database queries and the structure of statistical relational models [3].

As the space of possible definitions (e.g. all Datalog programs) is enormous, relational learning algorithms must employ heuristics to constraint the search space. These heuristics are generally specified through a *language bias*. One form of language bias is *syntactic bias*, which restricts the structure and syntax of the learned Datalog programs. Relational learning systems usually allow users to specify the syntactic bias through statements called *predicate definitions* and *mode definitions* [3]. Predicate and mode definitions express several types of restrictions on the structure of the learned Datalog programs, such as the relations allowed to be in the Datalog program, whether an attribute can appear as a variable or constant, and whether two relations can join. Table II shows a fragment of predicate and mode definitions used for learning the *advisedBy* relation over the UW-CSE database. A detailed explanation of these definitions is given in Section II. To the best of our knowledge, all (statistical) relational learning systems require some form of syntactic bias to restrict the hypothesis space.

student(stud)	professor(prof)
inPhase(stud,phase)	hasPosition(prof,position)
courseLevel(course,level)	taughtBy(course,prof,term)
ta(course,stud,term)	publication(title,author)

TABLE I
Schema fragments for the UW-CSE database.

Predicate definitions	Mode definitions
student(T1)	advisedBy(+,+)
inPhase(T1,T2)	student(+)
professor(T3)	inPhase(+,-)
hasPosition(T3,T4)	inPhase(+,#)
publication(T5,T1)	professor(+)
publication(T5,T3)	hasPosition(+,-)
...	

TABLE II
A subset of predicate and mode definitions for learning *advisedBy* relation.

For a relational learning algorithm to be effective and efficient, predicate and mode definitions must encode a great deal of information about the structure of the learned Datalog programs [3]. To set a sufficient degree of restriction, a user should know the internals of the learning algorithm and the schema of the input database, as well as having a relatively clear intuition on the structure of effective Datalog programs for the target relation. However, there may not be any user that knows both the database concepts, such as schema, and has a clear intuition about the target relation, particularly in specific domains such as biology. Hence, learning a relation requires many lengthy discussions between the database/machine learning expert and domain experts. Furthermore, the number of predicate and mode definitions is generally large and hard to debug and maintain. Users normally improve the initial set of definitions via trial and error, which is a tedious and time-consuming process. Hence, it takes a lot of time and effort to write and maintain these definitions, particularly for a relatively complex schema. In our conversations with relational learning experts, they have called predicate and mode definitions the “black magic” needed to make relational learning work and believe them to be a major reason for the relative unpopularity of these algorithms among users.

We demonstrate AutoMode, a system that leverages the information in the schema and content of the database to automatically generate predicate and mode definitions. We show that the predicate and mode definitions produced by AutoMode deliver the same accuracy as the manually written and tuned ones while imposing only a modest running-time overhead over large real-world databases.

II. Background

A. Relational Learning

An *atom* is a formula in the form of $R(e_1, \ldots, e_n)$, where R is a relation symbol. A *literal* is an atom, or the negation of an atom. Each attribute in a literal is set to either a variable or a constant, i.e., value. Variable and constants are also called *terms*. A *Horn clause* (clause for short) is a finite set of literals that contains exactly one positive literal. Horn clauses are also called conjunctive queries. A *Horn definition* is a set of Horn clauses with the same positive literal. A relational learning algorithm learns a Horn definition from input relational databases and training data. The learned definition is called the hypothesis. The *hypothesis space* is the set of all candidate Horn definitions.

Relational learning algorithms search over the hypothesis space to find a definition that covers as many positive examples as possible, while covering the fewest possible negative examples. Relational learning algorithms generally follow either a top-down or a bottom-up approach. Top-down algorithms start with an empty definition and iteratively add literals to the definition until the definition cannot be improved. Bottom-up algorithms first construct the most specific clause that covers a given positive example, and then generalize this clause to cover more positive examples.

B. Language Bias

In relational learning algorithms, language bias restricts the structure and syntax of the generated clauses. Language bias is specified through predicate and mode definitions [3].

Predicate definitions assign one or more *types* to each attribute in a database relation. In a candidate clause, two relations can be joined over two attributes (i.e., attributes are assigned the same variable) only if the attributes have the same type. For instance, in Table II, the predicate definition `student(T1)` indicates that the attribute in relation *student* is of type `T1`, and the predicate definition `inPhase(T1,T2)` indicates that the first and second attributes of relation *inPhase* are of type `T1` and `T2`, respectively. Therefore, relations *student* and *inPhase* can be joined on attributes *student[stud]* and *inPhase[stud]*. It is possible to assign multiple types to an attribute. For example the predicate definitions `publication(T5,T1)` and `publication(T5,T3)` indicate that the attribute *author* in relation *publication* belongs to both types `T1` and `T3`. Predicate definitions restrict the joins that can appear in a candidate clause: two relations can be joined only if their attributes share a type.

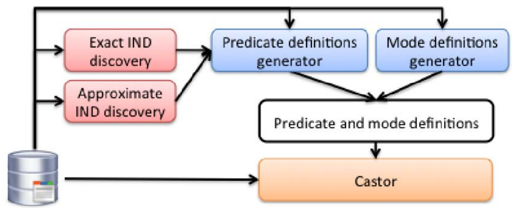

Exact IND discovery
Approximate IND discovery
Predicate definitions generator
Mode definitions generator
Predicate and mode definitions
Castor

Fig. 1. AutoMode is implemented on top of Castor.

Mode definitions indicate whether a term in an atom should be a new variable (i.e., existentially quantified variable), an existing variable (i.e., appears in a previously added atom), or a constant. They do so by assigning one or more symbols to each attribute in a relation. Symbol $+$ indicates that a term must be an existing variable, except for the atom in the head of a Horn clause. Symbol $-$ indicates that a term can be an existing variable or a new variable. For instance, the mode definition `inPhase(+,-)` in Table II indicates that the first term must be an existing variable and the second term can be either an existing or a new variable. Symbol $\#$ indicates that a term should be a constant. For instance, the mode definition `inPhase(+,#)` indicates that the second term must be a constant. Each atom in a candidate clause must satisfy at least one mode definition.

III. AutoMode System

Figure 1 shows the components of the AutoMode system. We implement AutoMode on top of Castor [4], a bottom-up relational learning system. AutoMode reads and extracts the information about the schema of the underlying database from the RDBMS. It then generates predicate and mode definitions in a pre-processing step. Castor uses these definitions to learn the definition of some target relation. The same predicate and mode definitions can be used to learn different target relations.

A. Generating Predicate Definitions

Let R and S be two relation symbols in the schema of the underlying database. Let $R(e_1, \cdots, e_n)$ and $S(o_1, \cdots, o_m)$ be two atoms in a clause C. Let e_i be the term in attribute $R[A]$ and o_j be the term in attribute $S[B]$, and let e_i and o_j be assigned the same variable or constant. That is, clause C joins R and S on A and B. Clause C is satisfiable only if these attributes share some values in the input database. Typically, the more frequently used joins are the ones over the attributes that participate in inclusion dependencies (INDs), such as foreign-key to primary-key referential constraints. AutoMode uses INDs in the input database to find which attributes, among all relations, share the same type. Let X and Y be sets of attribute names in R and S, respectively. Let I_R and I_S be the relations of R and S in the database. Relations I_R and I_S satisfy *exact IND* (*IND* for short) $R[X] \subseteq S[Y]$ if $\pi_X(I_R) \subseteq \pi_Y(I_S)$. If X and Y each contain only a single attribute, the IND is a *unary IND*. Given IND $R[X] \subseteq S[Y]$ in a database, the database satisfies unary IND $R[A] \subseteq S[B]$, where $A \in X$ and $B \in Y$. INDs are normally stored in the schema of the database. If they are not available in the schema, one can extract them from the database content. AutoMode uses the Binder algorithm [2] to discover INDs from the database, shown by the **Exact IND discovery** box in Figure 1, and generates all unary INDs implied by them.

We have observed that using exact INDs is not enough for generating helpful predicate definitions. For instance, to learn the accurate definition for relation *advisedBy* presented in Section I, Castor must join relations *publication*, *student*, and *professor* on attributes *publication[author]*, *student[stud]*, and *professor[prof]*. But, the UW-CSE database does not satisfy INDs *publication[author]* $\subseteq$ *student[stud]* or *publication[author]* $\subseteq$ *professor[prof]* because *publication[author]* contains both students and professors. Hence, AutoMode also uses *approximate INDs* to assign types to attributes. In an *approximate unary IND* $(R[A] \subseteq S[B], \alpha)$, one has to remove at least α fraction of the distinct values in $R[A]$ so that the database satisfies $R[A] \subseteq S[B]$ [2]. Approximate INDs are not usually maintained in schema and are discovered from the database content. We have implemented a program to extract approximate INDs from the database, shown by the **Approximate IND discovery** box in Figure 1. We use a relatively high error rate, 50%, for the approximate INDs to enlarge Castor hypothesis space.

After discovering unary exact and approximate INDs, AutoMode runs Algorithm 1 to generate a directed graph called *type graph*, which it then uses to assign types to attributes. First, it creates a graph whose nodes are attributes in the input schema and has an edge between each pair of attributes that participate in an exact or approximate IND (lines 1-3). Figure 2 shows an example of the type graph containing a subset of the attributes in the UW-CSE schema, where edges corresponding to exact and approximate INDs are shown by solid and dashed lines, respectively. If there are both approximate INDs $(R[A] \subseteq S[B], \alpha_1)$ and $(S[B] \subseteq R[A], \alpha_2)$, AutoMode uses only the one with lower error rate. The algorithm then assigns a new type to every node in the graph without any outgoing edges (lines 4-5). For example, it assigns new types `T1`, `T3`, and `T5` to *student[stud]*, *professor[prof]*, and *publication[title]*, respectively, in Figure 2. If there are cycles in the type graph, the algorithm assigns the same new type to all nodes in each cycle (lines 6-7). Next, it propagates the assigned type of each attribute to its neighbors in the reverse direction of edges in the graph until no changes are made to the graph (lines 8-11). For example, in Figure 2, the algorithm propagates type `T1` to *inPhase[stud]* and *ta[stud]* and attribute *publication[author]* inherits types `T1` and `T3` from *student[stud]* and *professor[prof]*, respectively. Because the error rates of approximate INDs accumulate over multiple edges in the graph, AutoMode propagates types only once over edges that correspond to approximate INDs.

Given the resulting graph, for each relation, AutoMode computes the Cartesian product of the types associated with its attributes. For each tuple in this Cartesian product, it produces a predicate definition for the relation. For instance, given the type assignment in Figure 2, AutoMode generates predicate definitions `publication(T5,T1)` and `publication(T5,T3)` for the *publication* relation.

Algorithm 1: Algorithm to generate the type graph.

```
   Input  : Schema $\mathcal{S}$ and all unary INDs $\Sigma$.
   Output: Type graph $G$.
1  create graph $G = (V, E)$ where $V$ contains a node for each
   attribute in the schema and $E = \varnothing$
2  foreach IND $R[A] \subseteq S[B] \in \Sigma$ do
3      add edge $v \to u$ to $E$, where $v$ and $u$ correspond to
       attributes $R[A]$ and $S[B]$, respectively
4  foreach node $u \in V$ without outgoing edges do
5      generate new type $T$ and set $types(u) = \{T\}$
6  foreach cycle $K \subseteq V$ do
7      generate new type $T$ and set $types(u) = \{T\}$ $\forall u \in K$
8  repeat
9      foreach $v \to u \in E$ where $types(u) \neq \varnothing$ do
10         set $types(v) = types(v) \cup types(u)$
11 until no changes in $G$
12 return $G$
```

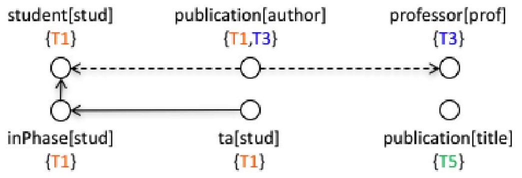


Fig. 2. A fragment of the type graph for UW-CSE database.

B. Generating Mode Definitions

AutoMode lets every attribute of every relation be a variable. However, it forces at least one variable in an atom to be an existing variable, i.e., appears in previously added atoms, to avoid generating Cartesian products in the clause. For each attribute A in relation R, AutoMode generates a mode definition for R where attribute A is assigned the $+$ symbol and all other attributes are assigned the $-$ symbol. If the number of distinct values in an attribute is below some given threshold, AutoMode allows the attribute to be a constant. This threshold is a hyper-parameter, but it has a relatively intuitive meaning. For each relation R in the database, AutoMode finds all attributes in R that can be constants using the aforementioned rule. Then, it computes the power set $\mathbf{M}$ of these attributes. For each non-empty set $M \in \mathbf{M}$, AutoMode generates a new set of mode definitions where it assigns $+$ and $-$ symbols as described above, except for the attributes in M, which are assigned the $\#$ symbol.

C. Empirical Results

Our empirical results indicate that AutoMode is generally as accurate as manual tuning. We learn the relation *advisedBy(stud,prof)* over the UW-CSE database, which is described in Section I and contains 1.8K tuples, 102 positive and 204 negative examples. We also learn relation *antiHIV(comp)*, which indicates that a chemical compound *comp* has anti-HIV activity, over the HIV database that contains structural information about chemical compounds (*wiki.nci.nih.gov/display/NCIDTPdata*). This database contains 80 relations with a total of 14M tuples, 2K positive and 4K negative examples. We try three other ways of setting the

Dataset	Measure	Baseline	Baseline (w/o const.)	Manual tuning	AutoMode
UW-CSE	F1-score	0.60	0.62	0.67	0.67
	Time	47s	6.6s	11s	10.8s
HIV	F1-score	-	0.80	0.83	0.84
	Time	>36h	20h	14.7m	32.2m

TABLE III
RESULTS OF LEARNING RELATIONS OVER UW-CSE AND HIV DATA (H=HOURS, M=MINUTES, S=SECONDS).

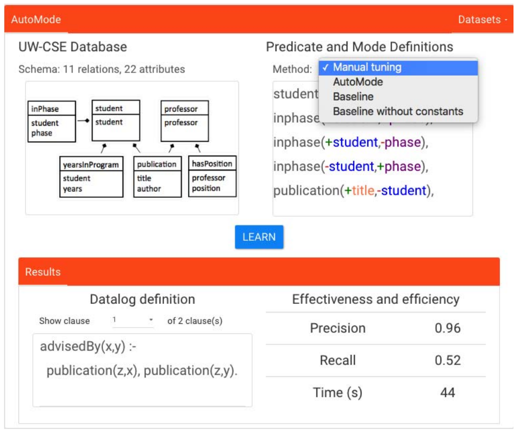
language bias. **Baseline** assigns the same types to all attributes and allows every attribute to be a variable or a constant. **Baseline without constants** is the same as the baseline method, except that it does not allow any attribute to be a constant. **Manual tuning** uses the language bias written by an expert. The expert had to learn the schema and go through several trial and error phases by running the underlying learning system and observing its results to write the predicate and mode definitions. The pre-processing step of AutoMode to extract INDs takes 2 seconds and 45 minutes over the UW-CSE and HIV databases, respectively. According to Table III, AutoMode is at least as accurate as manual tuning and more accurate than the baseline methods. Although learning using AutoMode takes longer than manual tuning, AutoMode eliminates the language bias preparation phase, which requires experts to spend hours defining and maintaining language bias. Also, AutoMode enables non-experts to use relational learning systems easily. We have performed experiments over other large databases, and have found that AutoMode allows Castor to learn accurate definitions efficiently [1].

IV. Demonstration Scenarios

In our demonstration, we walk the audience through the process of using a relational learning system to learn the definitions for the target relations over the UW-CSE, HIV, and IMDb (*imdb.com*) databases, and show how AutoMode simplifies this process. We also compare AutoMode against other methods of setting language bias for relational learning.

End-to-end learning system: The ultimate goal of AutoMode is to make relational learning systems easy to use. Therefore, we show an end-to-end scenario in which AutoMode is used along with Castor to learn the definition for a given relation. We provide an interface, shown in Figure 3, where the audience can select a database from a list of databases, e.g., UW-CSE, choose the tables containing positive and negative examples for the target relation, e.g., *advisedBy(stud,prof)*, and initiate the learning process. The system uses AutoMode to automatically generate predicate and mode definitions and uses them to learn a Datalog definition for the target relation. The system presents the Datalog definition and shows its accuracy and the learning time.

Impact of language bias: We manually prepare the predicate and mode definitions for several target concepts over UW-CSE, HIV, and IMDb databases. The audience will be presented with the schema of a database, e.g., UW-CSE, a target relation over the database, e.g., *advisedBy*, and a set of predicate and mode definitions for this target relation. We demonstrate the impact of language bias by modifying the



student

inphase

inphase(+student,-phase),

inphase(-student,+phase),

publication(+title,-student),

Datalog definition

advisedBy(x,y) :-
publication(z,x), publication(z,y).

Effectiveness and efficiency	
Precision	0.96
Recall	0.52
Time (s)	44

Fig. 3. AutoMode user interface.

manually written predicate and mode definitions, running the learning algorithm, and showing the learned definition and its accuracy. For instance, if we remove predicate definition `publication(T5,T3)` in Table II, Castor will not learn a definition containing the *publication* relation where the author is a professor, resulting in a less accurate definition. If time permits, the audience can also modify the predicate and mode definitions and explore its impact on learning.

How AutoMode works: We demonstrate how AutoMode generates predicate and mode definitions. We walk the audience through the discovery of the exact and approximate INDs over the database of their choice. Then, we present the audience with the *type graph* derived from this set of INDs as explained in Section III-A. Finally, we show the predicate and mode definitions generated from the type graph. To make these definitions easy to follow, we provide a visual representation of them in the form of a graph in which nodes represent attributes and there is an edge between two nodes if they share the same type, i.e., they can be joined.

Comparing AutoMode with other methods of setting language bias: We provide an interface, shown in Figure 3, where the audience can select from the four methods of setting the language bias described in Section III-C over the database of their choice from the list of available databases. After selecting a method, the system shows the predicate and mode definitions generated by this method. Then, the audience can run Castor using the selected method and observe the learned definition, its accuracy, and the learning time.

V. Acknowledgements

This work is supported by the National Science Foundation under grant IIS-1423238 and by AFRL Contract No. FA8750-16-C-0044 under the DARPA BRASS program.

References

[1] AutoMode: Technical Report. https://arxiv.org/abs/1710.01420, 2017.
[2] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: A survey. *The VLDB Journal*, 24(4), 2015.
[3] L. De Raedt. *Logical and Relational Learning*. Springer Publishing Company, Incorporated, 1st edition, 2010.
[4] J. Picado, A. Termehchy, A. Fern, and P. Ataei. Schema Independent Relational Learning. In *SIGMOD*, 2017.