Pruning Hypothesis Spaces Using Learned Domain Theories

Martin Svatoš[1], Gustav Šourek[1], Filip Železný[1], Steven Schockaert[2], and Ondřej Kuželka[2]

[1] Czech Technical University, Prague, Czech Republic
{svatoma1,souregus,zelezny}@fel.cvut.cz
[2] School of CS & Informatics, Cardiff University, Cardiff, UK
{SchockaertS1,KuzelkaO}@cardiff.ac.uk

Abstract. We present a method that uses learned domain theories to speed up hypothesis search in inductive logic programming. The key insight exploited in this work is that hypotheses can be equivalent, relative to some domain theory, even if they are not isomorphic. To detect such equivalences we use learned domain theories to saturate the hypotheses and then check if these saturations become isomorphic. We experimentally show that pruning equivalent hypotheses can lead to considerable reductions in both computation time and memory consumption.

1 Introduction

Typical ILP algorithms (e.g. Progol [10]) search through large hypothesis spaces. Methods for pruning this search space have the potential to lead to dramatic improvements in the quality of learned hypotheses, in the required runtime, or in both. One way of pruning the search space is by filtering isomorphic hypotheses, which is used, for instance, in the relational pattern mining algorithm Farmr [12]. However, pruning isomorphic hypotheses is often not optimal. Frequently we would want to prune hypotheses which are not isomorphic, but which are still equivalent, given some domain knowledge (while still maintaining completeness of the hypothesis search).

In this paper, we introduce a method which explicitly ties to prune equivalent hypotheses. One important challenge is that we need to be able to quickly test whether a new clause is equivalent to previously considered clauses. To this end, we develop a saturation method which, given a first-order-logic clause, can derive another longer *saturated* clause which is equivalent to it modulo a domain theory. The saturation method has the important property that two clauses are equivalent, given a domain theory, if and only if their saturations are isomorphic.

As we demonstrate experimentally, the use of saturations can be orders of magnitude more effective than methods that merely check for isomorphism. We also present a method for learning the necessary domain theories from data. We show that our method leads to significant speed-ups, even when taking into account the time that was needed for learning the domain theories.

2 Preliminaries

In this section, we first give an overview of the notations and terminology from first-order logic that will be used throughout the paper, after which Section 2.2 describes the considered learning setting.

2.1 First-Order Logic

We consider a function-free first-order language, which is built from a finite set of constants, variables, and predicates in the usual way. A term is a variable or a constant. An atom is a formula of the form $p(t_1, ..., t_n)$, where p is an n-ary predicate symbol and $t_1, ..., t_n$ are terms. A literal is an atom or the negation of an atom. A clause A is a universally quantified disjunction of literals $\forall x_1 ... \forall x_n$: $\phi_1 \vee ... \vee \phi_k$, such that $x_1, .., x_n$ are the only variables occurring in the literals $\phi_1, ..., \phi_k$. For the ease of presentation, we will sometimes identify a clause A with the corresponding set of literals $\{\phi_1, ..., \phi_k\}$. The set of variables occurring in a clause A is written as *vars*(A) and the set of all terms as *terms*(A). For a clause A, we define the *sign flipping* operation as $\widetilde{A} \stackrel{def}{=} \bigvee_{l \in A} \tilde{l}$, where $\tilde{a} = \neg a$ and $\tilde{\neg a} = a$ for an atom a. In other words, the sign flipping operation simply replaces each literal by its negation.

A substitution θ is a mapping from variables to terms. For a clause A, we write $A\theta$ for the clause $\{\phi\theta \,|\, \phi \in A\}$, where $\phi\theta$ is obtained by replacing each occurrence in ϕ of a variable v by the corresponding term $\theta(v)$. If A and B are clauses then we say that A *θ-subsumes* B (denoted $A \preceq_\theta B$) if and only if there is a substitution θ such that $A\theta \subseteq B$. If $A \preceq_\theta B$ and $B \preceq_\theta A$, we call A and B *θ-equivalent* (denoted $A \approx_\theta B$). Note that the $\approx_\theta$ relation is indeed an equivalence relation (i.e. it is reflexive, symmetric and transitive). Clauses A and B are said to be *isomorphic* (denoted $A \approx_{iso} B$) if there exists an injective substitution θ such that $A\theta = B$.

Example 1. Let us consider the following four clauses:

$$C_1 = p_1(A, B) \vee \neg p_2(A, B)$$
$$C_2 = p_1(A, B) \vee \neg p_2(A, B) \vee \neg p_2(A, C)$$
$$C_3 = p_1(X, Y) \vee \neg p_2(X, Y) \vee \neg p_2(X, Z)$$
$$C_4 = p_1(A, B) \vee \neg p_3(A, B)$$

Then we can verify easily that $C_1 \approx_\theta C_2 \approx_\theta C_3$ (hence also $C_i \preceq_\theta C_j$ for $i, j \in \{1, 2, 3\}$). We also have $C_1 \not\approx_{iso} C_2$, $C_1 \not\approx_{iso} C_3$ and $C_2 \approx_{iso} C_3$. Finally $C_i \not\preceq_\theta C_4$ and $C_4 \not\preceq_\theta C_i$ for any $i \in \{1, 2, 3\}$. The clauses C_2 and C_3 are θ-reducible.

A literal is ground if it does not contain any variables. A grounding substitution is a substitution in which each variable is mapped to a constant. Clearly, if θ is a grounding substitution, then for any literal ϕ it holds that $\phi\theta$ is ground. An interpretation ω is defined as a set of ground literals. A clause $A = \{\phi_1, ..., \phi_n\}$

is satisfied by ω, written $\omega \models A$, if for each grounding substitution θ, it holds that $\{\phi_1\theta, ..., \phi_n\theta\} \cap \omega \neq \emptyset$. In particular, note that a ground literal ϕ is satisfied by ω if $\phi \in \omega$. The satisfaction relation $\models$ is extended to (sets of) propositional combinations of clauses in the usual way. If $\omega \models T$, for T a propositional combination of clauses, we say that ω is a model of T. If T has at least one model, we say T is satisfiable. Finally, for two (propositional combinations of) clauses A and B, we write $A \models B$ if every model of A is also a model of B. Note that if $A \preceq_\theta B$ for clauses A and B then $A \models B$, but the converse does not hold in general.

Deciding θ-subsumption between two clauses is an NP-complete problem. It is closely related to constraint satisfaction problems with finite domains and tabular constraints [3], conjunctive query containment [2] and homomorphism of relational structures. The formulation of θ-subsumption as a constraint satisfaction problem has been exploited in the ILP literature for the development of fast θ-subsumption algorithms [9, 7]. CSP solvers can also be used to check whether two clauses are isomorphic, by using the primal CSP encoding described in [9] together with an *alldifferent* constraint [5] over CSP variables representing logical variables. In practice, this approach to isomorphism checking can be further optimized by pre-computing a directed hypergraph variant of Weisfeiler-Lehman coloring [17] (where terms play the role of hyper-vertices and literals the role of directed hyper-edges) and by enriching the respective clauses by unary literals with predicates representing the labels obtained by the Weisfeiler-Lehman procedure, which helps the CSP solver to reduce its search space.

2.2 Learning Setting

In this paper we will work in the classical setting of *learning from interpretations* [13]. In this setting, examples are interpretations and hypotheses are clausal theories (i.e. conjunctions of clauses). An example e is said to be *covered* by a hypothesis H if $e \models H$ (i.e. e is *covered* by H if it is a model of H). Given a set of positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$, the training task is then to find a hypothesis H from some class of hypotheses $\mathcal{H}$ which optimizes a given scoring function (e.g. training error). For the ease of presentation, we will restrict ourselves to classes $\mathcal{H}$ of hypotheses in the form of clausal theories without constants, as constants can be emulated by unary predicates (since we do not consider functions).

The covering relation $e \models H$ can be checked using a θ-subsumption solver as follows. Each hypothesis H can be written as conjunction of clauses $H = C_1 \wedge \cdots \wedge C_n$. Clearly, $e \not\models H$ if there is an i in $\{1, \ldots, n\}$ such that $e \models \neg C_i$, which holds precisely when $C_i \preceq_\theta \neg(\bigwedge e)$.

Example 2. Let us have the example (inspired by the Michalski's East-West trains datasets [16])

$$e = \{eastBound(car1), hasCar(car1), hasLoad(car1, load1), boxShape(load1), \neg eastBound(load1), \neg hasCar(load1), \neg hasLoad(load1, car1), \neg hasLoad(load1, load1), \neg hasLoad(car1, car1), \neg boxShape(car1)\}$$

and two hypotheses H_1 and H_2

$$H_1 = eastBound(C) \vee \neg hasLoad(C, L) \vee \neg boxShape(L)$$
$$H_2 = \neg eastBound(C) \vee \neg hasLoad(C, L)$$

Then to check if $e \models H_i$, $i = 1, 2$, using a θ-subsumption solver, we construct

$$\neg(\bigwedge e) = \neg eastBound(car1) \vee \neg hasCar(car1) \vee \neg hasLoad(car1, load1) \vee \vee boxShape(load1) \vee eastBound(load1) \vee hasCar(load1) \vee \vee hasLoad(load1, car1) \vee hasLoad(load1, load1) \vee hasLoad(car1, car1) \vee boxShape(car1)$$

It is then easy to check that $H_1 \not\preceq_\theta \neg(\bigwedge e)$ and $H_2 \preceq_\theta \neg(\bigwedge e)$, from which it follows that $e \models H_1$ and $e \not\models H_2$.

In practice, when using a θ-subsumption solver to check $C_i \preceq_\theta \neg(\bigwedge e)$, it is usually beneficial to flip the signs of all the literals, i.e. to instead check $\widetilde{C_i} \preceq_\theta \bigvee e$, which is clearly equivalent. This is because θ-subsumption solvers often represent negative literals in interpretations implicitly to avoid excessive memory consumption[3], relying on the assumption that most predicates in real-life datasets are sparse.

2.3 Theorem Proving Using SAT Solvers

The methods described in this paper will require access to an efficient theorem prover for clausal theories. Since we restrict ourselves to function-free theories without equality, we can rely on a simple theorem-proving procedure based on propositionalization, which is a consequence of the following well-known result[4] [11].

Theorem 1 (Herbrand's Theorem). *Let $\mathcal{L}$ be a first-order language without equality and with at least one constant symbol, and let $\mathcal{T}$ be a set of clauses. Then $\mathcal{T}$ is unsatisfiable iff there exists some finite set $\mathcal{T}_0$ of $\mathcal{L}$-ground instances of clauses from $\mathcal{T}$ that is unsatisfiable.*

[3] This is true for the θ-subsumption solver based on [7] which we use in our implementation.

[4] The formulation of Hebrand's theorem used here is taken from notes by Cook and Pitassi: `http://www.cs.toronto.edu/~toni/Courses/438/Mynotes/page39.pdf`.

Here $A\theta$ is called an $\mathcal{L}$-ground instance of a clause A if θ is a grounding substitution that maps each variable occurring in A to a constant from the language $\mathcal{L}$.

In particular, to decide if $\mathcal{T} \models C$ holds, where T is a set of clauses and C is a clause (without constants and function symbols), we need to check if $\mathcal{T} \wedge \neg C$ is unsatisfiable. Since Skolemization preserves satisfiability, this is the case iff $\mathcal{T} \wedge \neg C_{Sk}$ is unsatisfiable, where $\neg C_{Sk}$ is obtained from $\neg C$ using Skolemization. Let us now consider the restriction $\mathcal{L}_{Sk}$ of the considered first-order language $\mathcal{L}$ to the constants appearing in C_{Sk}, or to some auxiliary constant s_0 if there are no constants in C_{Sk}. From Herbrand's theorem, we know that $\mathcal{T} \wedge \neg C_{Sk}$ is unsatisfiable in $\mathcal{L}_{Sk}$ iff the grounding of this formula, w.r.t. the constants from $\mathcal{L}_{Sk}$ is satisfiable, which we can efficiently check using a SAT solver. Moreover, it is easy to see that $\mathcal{T} \wedge \neg C_{Sk}$ is unsatisfiable in $\mathcal{L}_{Sk}$ iff this formula is unsatisfiable in $\mathcal{L}$ [5].

In practice, it is not always necessary to completely ground the formula $\mathcal{T} \wedge \neg C_{Sk}$. It is often beneficial to use an incremental grounding strategy similar to cutting plane inference in Markov logic [15]. To check if a clausal theory $\mathcal{T}$ is satisfiable, this method proceeds as follows.

Step 0: start with an empty Herbrand interpretation $\mathcal{H}$ and an empty set of ground formulas $\mathcal{G}$.

Step 1: check which groundings of the formulas in $\mathcal{T}$ are not satisfied by $\mathcal{H}$ (e.g. using a CSP solver). If there are no such groundings, the algorithm returns $\mathcal{H}$, which is a model of $\mathcal{T}$. Otherwise the groundings are added to $\mathcal{G}$.

Step 2: use a SAT solver to find a model of $\mathcal{G}$. If $\mathcal{G}$ does not have any model then $\mathcal{T}$ is unsatisfiable and the method finishes. Otherwise replace $\mathcal{H}$ by this model and go back to Step 1.

3 Pruning Hypothesis Spaces Using Domain Theories

In this section we show how domain theories can be used to prune the search space of ILP systems. Let us start with two motivating examples.

Example 3. Let us consider the following two hypotheses for some target concept x:

$$H_1 = x(A) \vee \neg animal(A) \vee \neg cod(A)$$
$$H_2 = x(A) \vee \neg fish(A) \vee \neg cod(A)$$

Intuitively, these two hypotheses are equivalent since every *cod* is a *fish* and and every *fish* is an *animal*. Yet ILP systems would need to consider both of

[5] Indeed, if $\mathcal{T} \wedge \neg C_{Sk}$ is unsatisfiable in $\mathcal{L}$, then there is a set of corresponding $\mathcal{L}$-ground instances of clauses that are unsatisfiable. If we replace each constant appearing in these ground clauses which does not appear in C_{sk} by an arbitrary constant that does appear in C_{sk}, then the resulting set of ground clauses must still be inconsistent, since T does not contain any constants and there is no equality in the language, meaning that $\mathcal{T} \wedge \neg C_{Sk}$ cannot be satisfiable in $\mathcal{L}_{Sk}$.

these hypotheses separately because H_1 and H_2 are not isomorphic, they are not θ-equivalent and neither of them θ-subsumes the other.

Example 4. Problems with redundant hypotheses abound in datasets of molecules, which are widespread in the ILP literature. For instance, consider the following two hypotheses:

$$H_1 = x(A) \vee \neg carb(A) \vee \neg bond(A,B) \vee \neg bond(B,C) \vee \neg hydro(C)$$
$$H_2 = x(A) \vee \neg carb(A) \vee \neg bond(A,B) \vee \neg bond(C,B) \vee \neg hydro(C)$$

These two hypotheses intuitively represent the same molecular structures (a carbon and a hydrogen both connected to the same atom of unspecified type). Again, however, their equivalence cannot be detected without the domain knowledge asserting that bonds in molecular datasets are symmetric[6].

In the remainder of this section we will describe how background knowledge can be used to detect equivalent hypotheses. First, we introduce the notion of *saturations* of clauses in Section 3.1. In Section 3.2, we then explain how these saturations can be used to efficiently prune search spaces of ILP algorithms. Finally, in Section 3.3 we describe a simple method for learning domain theories from the given training data.

3.1 Saturations

The main technical ingredient of the proposed method is the following notion of *saturation*.

Definition 1 (Saturation of a clause). *Let $\mathcal{B}$ be a clausal theory and C a clause (without constants or function symbols). If $\mathcal{B} \not\models C$, we define the saturation of C w.r.t. $\mathcal{B}$ to be the maximal clause C' satisfying: (i) $vars(C') = vars(C)$ and (ii) $\mathcal{B} \wedge C'\theta \models C\theta$ for any injective grounding substitution θ. If $\mathcal{B} \models C$, we define the saturation of C w.r.t. $\mathcal{B}$ to be $\mathbf{T}$ (tautology).*

When $\mathcal{B}$ is clear from the context, we will simply refer to C' as the saturation of C.

Definition 1 naturally leads to a straightforward procedure for computing the saturation of a given clause. Let $\mathcal{P} = \{l_1, l_2, \ldots, l_n\}$ be the set of all literals which can be constructed using variables from C and predicate symbols from $\mathcal{B}$ and C. Let θ be an arbitrary injective grounding substitution; note that we can indeed take θ to be arbitrary because $\mathcal{B}$ and C do not contain constants. If $\mathcal{B} \not\models C$, the saturation of C is given by the following clause:

$$\bigvee \{ l \in \mathcal{P} : \mathcal{B} \models \neg l\theta \vee C\theta \} \quad (1)$$

[6] In the physical world, bonds do not necessarily have to be symmetric, e.g. there is an obvious asymmetry in polar bonds. However, it is a common simplification in data mining on molecular datasets to assume that bonds are symmetric.

This means in particular that we can straightforwardly use the SAT based theorem proving method from Section 2.3 to compute saturations. The fact that (1) correctly characterizes the saturation can be seen as follows. If C' is the saturation of C then $\mathcal{B} \wedge C'\theta \models C\theta$ by definition, which is equivalent to $\mathcal{B} \wedge \neg(C\theta) \models \neg(C'\theta)$. We have $\neg(C'\theta) = \bigwedge\{\tilde{l}\theta : \mathcal{B} \wedge \neg(C\theta) \models \tilde{l}\theta\} = \bigwedge\{\tilde{l}\theta : \mathcal{B} \wedge l\theta \models C\theta\}$, and thus $C'\theta = \bigvee\{l\theta : \mathcal{B} \wedge l\theta \models C\theta\}$. Finally, since θ is injective, we have[7] $C' = (C'\theta)\theta^{-} = \bigvee\{l : \mathcal{B} \wedge l\theta \models C\theta\}$.

The next proposition will become important later in the paper as it will allow us to replace clauses by their saturations when learning from interpretations.

Proposition 1. *If C is a saturation of C' w.r.t. $\mathcal{B}$ then $\mathcal{B} \wedge C' \models C$.*

Proof. We have $\mathcal{B} \wedge C' \models C$ iff $\mathcal{B} \wedge C' \wedge \neg C$ is unsatisfiable. Skolemizing $\neg C$, this is equivalent to $\mathcal{B} \wedge C' \wedge \neg(C\theta_{Sk})$ being unsatisfiable, where θ_{Sk} is a substitution representing the Skolemization. As in Section 2.3, we find that the satisfiability of $\mathcal{B} \wedge C' \wedge \neg(C\theta_{Sk})$ is also equivalent to the satisfiability of the grounding of $\mathcal{B} \wedge C' \wedge \neg(C\theta_{Sk})$ w.r.t. the Skolem constants introduced by θ_{Sk}. In particular, this grounding must contain the ground clause $C'\theta_{Sk}$. From the definition of saturation, we have that $\mathcal{B} \wedge C'\theta_{Sk} \wedge \neg(C\theta_{Sk}) \models \mathbf{F}$[8] (recall that θ_{Sk} is injective and therefore we can use the property from the definition), and thus $\mathcal{B} \wedge C' \wedge \neg C \models \mathbf{F}$.

The next proposition shows that saturations cover the same examples as the clauses from which they were obtained, when $\mathcal{B}$ is a domain theory that is valid for all examples in the dataset.

Proposition 2. *Let $\mathcal{B}$ be a clausal theory such that for all examples e from a given dataset it holds that $e \models \mathcal{B}$. Let C be a clause and let C' be its saturation w.r.t. $\mathcal{B}$. Then for any example e from the dataset we have $(e \models C) \Leftrightarrow (e \models C')$.*

Proof. From the characterization of saturation in (1), it straightforwardly follows that $C \models C'$, hence $e \models C$ implies $e \models C'$. Conversely, if $e \models C'$, then we have $e \models \mathcal{B} \wedge C'$, since we assumed that $e \models \mathcal{B}$. Since we furthermore know from Proposition 1 that $\mathcal{B} \wedge C' \models C$, it follows that $e \models C$.

Finally, we define *positive* and *negative saturations*, which only add positive or negative literals to clauses. Among others, this will be useful in settings where we are only learning Horn clauses.

Definition 2. *A positive (resp. negative) saturation of C is defined as $C'' = C \cup \{l \in C' : l$ is a positive (resp. negative) literal$\}$ where C' is a saturation of C.*

Propositions 1 and 2 are also valid for positive or negative saturations; their proofs can be straightforwardly adapted. When computing the positive (resp. negative) saturation, we can restrict the set $\mathcal{P}$ of candidate literals to the positive (resp. negative) ones. This can speed up the computation of saturations significantly.

[7] Note that we are slightly abusing notation here, as θ^{-1} is not a substitution.

[8] Here, $\mathbf{F}$ denotes falsity.

3.2 Using Saturations for Pruning

The concept of saturation that was introduced in the previous section allows us to efficiently prune hypothesis spaces in ILP. Most ILP algorithms maintain some queue of candidate clauses. This is the case, for instance, in algorithms based on best-first search (Progol, Aleph [10]). Other algorithms are based on level-wise search, maintaining similar data structures (e.g. Warmr [4]). Many of the clauses that are stored in such queues or similar data structures will be equivalent, even if they are not isomorphic. Existing methods, even if they were removing isomorphic clauses during search[9], have to consider each of these equivalent clauses separately, which may greatly affect their performance.

To efficiently detect equivalences by checking isomorphism of saturations, we replace the queue data structure (or a similar data structure used by the given algorithm) by a data structure that is based on hash sets. When a new hypothesis H is constructed by the algorithm, we first compute its saturation H'. Then, we check whether the modified queue data structure already contains a clause that is isomorphic to H'. To efficiently check this, we use a straightforward generalization of the Weisfeiler-Lehman labeling procedure [17]. We then only need to check whether two clauses are isomorphic if they have the same hash value. We similarly check whether H' is isomorphic to a clause in the so-called closed set of previously processed hypotheses. If H' is neither isomorphic to a clause in the queue nor to a clause in the closed set, it is added to the queue.

In addition to equivalence testing, saturations can be used to filter trivial hypotheses, i.e. hypotheses covering every example, without explicitly computing their coverage on the dataset (which can be very costly on large datasets). We illustrate this use of saturations in the next example.

Example 5. Consider a domain theory $\mathcal{B} = \neg \mathit{professor}(X) \vee \neg \mathit{student}(X)$ which states that no one can be both a student and a professor. Let us also consider a hypothesis $H = \mathit{employee}(X) \vee \neg \mathit{professor}(X) \vee \neg \mathit{student}(X)$. If the domain theory $\mathcal{B}$ is correct, H should cover all examples from the dataset and is thus trivial. Accordingly, the saturation of H contains every literal, and is in particular equivalent to $\mathbf{T}$.

Finally, we note that it can be shown straightforwardly that the pruning method based on saturations not affect completeness of the hypothesis search methods considered here[10].

3.3 Learning Domain Theories for Pruning

The domain theories that we want to use for pruning hypothesis spaces can be learned from the given training dataset. Every clause C in such a learned domain

[9] For instance, Farmr [12] or RelF [8] remove isomorphic clauses (or conjunctive patterns), but many existing ILP systems do not attempt removing isomorphic clauses.

[10] We omit a formal proof due to limited space. However, note that preserving of completeness follows, first, from monotonicity of deduction in classical logic and, second, from the fact that we use isomorphism for pruning and not θ-equivalence.

theory should satisfy $e \models C$ for all examples e in the dataset. We construct such theories using a level-wise search procedure, starting with an empty domain theory. The level-wise procedure maintains a list of candidate clauses (modulo isomorphism) with i literals. If a clause C in the list of candidate clauses covers all examples (i.e. $e \models C$ for all e from the dataset) then it is removed from the list and if there is no clause in the domain theory which θ-subsumes C, then C is also added to the domain theory. Each of the remaining clauses in the list, i.e. those which do not cover all examples in the dataset, are then extended in all possible ways by addition of a literal. This is repeated until a threshold on the maximum number of literals is reached. The covering of examples by the candidate clauses is checked using θ-subsumption as outlined in Section 2.

It is worth pointing out that if we restrict the domain theories, e.g. to contain only clauses of length at most 2 or only Horn clauses, the saturation process will be guaranteed to run in polynomial time (which follows from polynomial-time solvability of 2-SAT and Horn-SAT).

4 Experiments

In this section we evaluate the usefulness of the proposed pruning method on real datasets. We test it inside an exhaustive feature construction algorithm which we then evaluate on a standard molecular datasets from the NCI GI 50 dataset collection [14].

4.1 Methodology and Implementation

The evaluated feature construction method is a simple level-wise algorithm which works similarly to the Warmr frequent pattern mining algorithm [4]. It takes two parameters: maximum depth d and maximum number of covered examples t (also called "maximum frequency"). It returns all connected[11] clauses which can be obtained by saturating clauses containing at most d literals, and which cover at most t examples. Unlike in frequent conjunctive pattern mining where minimum frequency constraints are natural, when mining in the setting of learning from interpretations, the analogue of the minimum frequency is the maximum frequency constraint[12].

The level-wise algorithm expects as input a list of interpretations (examples) and the parameters t and $d > 0$. It proceeds as follows:

Step 0: set $i := 0$ and $L_0 := \{\square\}$ where $\square$ denotes the empty clause.

[11] A clause is said to be connected if it cannot be written as disjunction of two non-empty clauses. For instance $\forall X, Y : p_1(X) \vee p_2(Y)$ is not connected because it can be written also as $(\forall X : p_1(X)) \vee (\forall Y : p_2(Y))$ but $\forall X, Y : p_1(X) \vee p_2(Y) \vee p_3(X, Y)$ is connected. If a clause is connected then its saturation is also connected.

[12] Frequent conjunctive pattern mining can be emulated in our setting. It is enough to notice that the clauses that we construct in our setting are just negations of conjunctive patterns.

Step 1: construct a set L_{i+1} by extending each clause from L_i with a negative literal (in all possible ways).

Step 2: replace clauses in L_{i+1} by their negative saturations and for each set of mutually isomorphic clauses keep only one of them.

Step 3: remove from L_{i+1} all clauses which cover more than t examples in the dataset.

Step 4: if L_{i+1} is empty or $i+1 > d$ then finish and return $\bigcup_{j=0}^{i+1} L_j$. Otherwise set $i := i+1$ and go to step 1.

As can be seen from the above pseudocode, we restricted ourselves to mining clauses which contain only negative literals. This essentially corresponds to mining positive conjunctive queries, which is arguably the most typical scenario. Nonetheless, it would be easy to allow the algorithm to search for general clauses (as the θ-subsumption solver used in the implementation actually allows efficient handling of negations).

We implemented the level-wise algorithm and the domain theory learner in Java[13]. For checking coverage of examples using θ-subsumption, we used an implementation of the θ-subsumption algorithm from [7]. For theorem proving, we used an incremental grounding solver[14] which relies on the Sat4j library [1] for solving ground theories and the θ-subsumption engine from [7].

4.2 Results

We measured runtime and the total number of clauses returned by the level-wise algorithm without saturations and with saturations. Both algorithms were exactly the same, the only difference being that the second algorithm first learned a domain theory and then used it for computing the saturations. Note in particular that both algorithms used the same isomorphism filtering. Therefore any differences in computation time must be directly due to the use of saturations.

We performed the experiments reported here on the NCI dataset KM20L2. The learned domain theories were restricted to contain only clauses with at most two literals. We set the maximum number of covered examples equal to the number of examples in the dataset minus one (which corresponds to a minimum frequency constraint of 1 when we view the clauses as negated conjunctive patterns). Then we also performed an experiment where we set it equal to the number of examples in the dataset minus 50 (which analogically corresponds to minimum frequency constraint 50). We set the maximum time limit to 2 hours.

The results of the experiments are shown in Figure 1. The pruning method based on saturations turns out to pay off when searching for longer clauses where it improves the baseline by more than an order of magnitude. When searching for smaller clauses, the runtime is dominated by the time for learning the domain theory, which is why the baseline algorithm is faster in that case. The number

[13] Available from https://github.com/martinsvat.

[14] Available from https://github.com/supertweety and originally developed for inference in possibilistic logic [6].

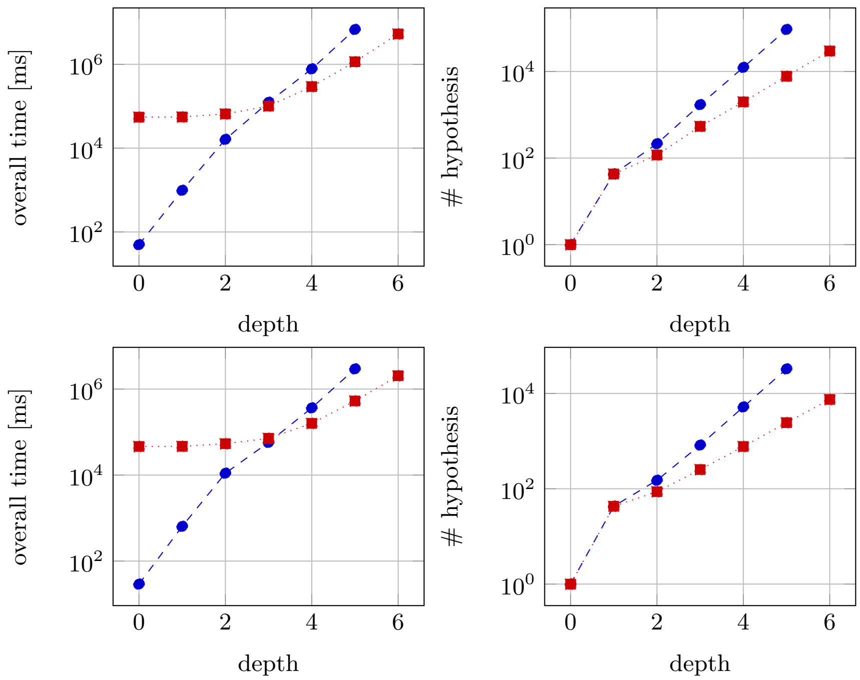

Fig. 1. Left panels: Runtime of the level-wise algorithm using saturations for pruning (red) and without using saturations (blue). **Right panels:** Number of clauses constructed by the algorithm using saturations (red) and without using saturations (blue). Top panels display results for maximal number of covered examples equal to dataset size minus one and bottom panels for this parameter set to dataset size minus 50.

of generated clauses, which is directly proportional to memory consumption, also becomes orders of magnitude smaller when using saturations for longer clauses. Note that for every clause constructed by the baseline algorithm, there is an equivalent clause constructed by the algorithm with the saturation-based pruning. We believe these results clearly suggest the usefulness of the proposed method, which could be also used inside many existing ILP systems.

5 Conclusions

In this paper, we introduced a generally applicable method for pruning hypotheses in ILP, which goes beyond mere isomorphism testing. We showed that the method is able to reduce the size of the hypothesis space by orders of magnitudes, and also leads to a significant runtime reduction. An interesting aspect of the proposed method is that it combines induction (domain theory learning) and deduction (theorem proving) for pruning the search space. In future, it would be interesting to combine these two components of our approach more tightly.

Acknowledgements MS, GŠ and FŽ acknowledge support by project no. 17-26999S granted by the Czech Science Foundation. OK is supported by a grant from the Leverhulme Trust (RPG-2014-164). SS is supported by ERC Starting Grant 637277. Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

References

1. Berre, D.L., Parrain, A.: The SAT4J library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation **7**, 50–64 (2010)
2. Chekuri, C., Rajaraman, A.: Conjunctive query containment revisited. Theor. Comput. Sci. **239**(2), 211–229 (2000)
3. Dechter, R.: Constraint processing. Elsevier Morgan Kaufmann (2003)
4. Dehaspe, L., Raedt, L.D.: Mining association rules in multiple relations. In: Inductive Logic Programming, 7th International Workshop, ILP-97, Prague, Czech Republic, September 17-20, 1997, Proceedings, pp. 125–132 (1997)
5. van Hoeve, W.J.: The alldifferent constraint: A survey. CoRR **cs.PL/0105015** (2001). URL http://arxiv.org/abs/cs.PL/0105015
6. Kuželka, O., Davis, J., Schockaert, S.: Encoding markov logic networks in possibilistic logic. In: 31st Conference on Uncertainty in Artificial Intelligence, UAI 2015, pp. 454–463 (2015)
7. Kuželka, O., Železný, F.: A restarted strategy for efficient subsumption testing. Fundam. Inform. **89**(1), 95–109 (2008)
8. Kuželka, O., Železný, F.: Block-wise construction of tree-like relational features with monotone reducibility and redundancy. Mach. Learn. **83**(2), 163–192 (2011)
9. Maloberti, J., Sebag, M.: Fast theta-subsumption with constraint satisfaction algorithms. Machine Learning **55**(2), 137–174 (2004)
10. Muggleton, S.: Inverse entailment and progol. New Gen. Comput. **13**(3&4), 245–286 (1995)
11. Newborn, M.: Automated theorem proving - theory and practice. Springer (2001)
12. Nijssen, S., Kok, J.N.: Efficient frequent query discovery in FARMER. In: Knowledge Discovery in Databases: PKDD'03, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, pp. 350–362 (2003)
13. Raedt, L.D.: Logical settings for concept-learning. Artif. Intell. **95**(1), 187–201 (1997)
14. Ralaivola, L., Swamidass, S.J., Saigo, H., Baldi, P.: Graph kernels for chemical informatics. Neural Netw. **18**(8), 1093–1110 (2005)
15. Riedel, S.: Improving the accuracy and efficiency of MAP inference for markov logic. In: UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial, pp. 468–475 (2008)
16. Stepp, R.E., Michalski, R.S.: Conceptual clustering: Inventing goal-oriented classifications of structured objects. Machine learning: An artificial intelligence approach **2**, 471–498 (1986)
17. Weisfeiler, B., Lehman, A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia **2**(9), 12–16 (1968)