Reducing Examples in Relational Learning with Bounded-Treewidth Hypotheses

Ondřej Kuželka, Andrea Szabóová, and Filip Železný

Faculty of Electrical Engineering, Czech Technical University in Prague
Technicka 2, 16627 Prague, Czech Republic
{kuzelon2,szaboand,zelezny}@fel.cvut.cz,

Abstract. We study reducibility of learning examples in the learning from entailment setting. We start with an existing reduction method and improve it for the case when learned hypotheses are restricted to have bounded treewidth. We show that in such cases there is a polynomial-time reduction algorithm which reduces the learning examples at least as much as the existing exponential-time algorithm despite the fact that the examples which are reduced can have arbitrarily high treewidth.

1 Introduction

In this paper, we study the question how much can relational learning examples be reduced while guaranteeing that the set of logical formulas which can be induced from these examples would not be affected. This problem has already been studied in the case when there were no restrictions on the structure of the learnt logical formulas [1]. Here, we study also the case when we know that the formulas to be learnt have bounded treewidth or are acyclic. We show that in this case, interestingly, learning examples can be reduced in polynomial time, and moreover, that they can be reduced at least as much as in the formerly studied unrestricted-structure case. The proposed method is based on application of k-consistency algorithm [2] known from constraint satisfaction. We show that k-consistency is sufficiently strong to check whether learning examples with arbitrary treewidth parameter are equivalent when hypotheses are guaranteed to have treewidth bounded by k. Normally, k-consistency can decide equivalence only for examples with treewidth bounded by k but here we use it for examples with arbitrarily high treewidth. And indeed, the examples which are equivalent for learning of formulas with treewidth at most k may be non-equivalent for formulas with higher treewidth.

2 Illustration of the Approach: Learning with Graphs

We illustrate our method for reduction of learning examples on the problem of learning with labelled graphs. Let us have two sets of labelled graphs (with labelled vertices and edges) - positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$ sampled from some distribution. The basic task is to find a set of labelled graphs

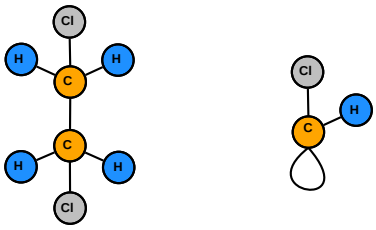

Fig. 1. An example of a graph representing a molecule (left) and another graph which is indistinguishable from it using pattern graphs from the set of labelled trees (right).

(*patterns*) that are able to classify the learning examples and predict the classification of unseen graphs as precisely as possible. Given a set of pattern graphs $\mathcal{P} = \{P_1, \ldots, P_k\}$ (i.e. a *hypothesis*), a graph G is classified as positive example if there is at least one pattern graph $P \in \mathcal{P}$ for which there exists a homomorphism to G. Deciding if there is a homomorphism between two labelled graphs is NP-complete. The set of pattern graphs is usually searched using a heuristic algorithm. The heuristic employed for searching usually requires computation of homomorphisms between the candidate pattern graphs and the graphs in the datasets. The NP-complete homomorphism problem then often becomes a bottleneck of such methods. Speeding it up is thus an important problem. In this paper, we develop methods that allow us to reduce size of learning examples in problems including graph learning which, in turn, means faster homomorphism checking. We base our approach on the observation that when the set of hypotheses (pattern graphs in this case) is limited then some examples (graphs) might be indistinguishable by hypotheses from this set. We can therefore replace every graph in the set of learning examples by the smallest graph indistinguishable from it. An example of a graph representing a molecule and another graph which is indistinguishable from it using pattern graphs from the set of labelled trees are shown in Figure 1. Rather than limiting ourselves to labelled graphs, we base our approach on first-order logic and develop a method for reduction of learning examples in the form of first-order-logic clauses, which can be used generally in relational learning including graph learning problems.

3 Preliminaries: Logic, Constraint Satisfaction, Treewidth

Let us first state some notational conventions. A first-order-logic clause is a universally quantified disjunction of first-order-logic literals. For convenience, we do not write the universal quantifiers explicitly. We treat clauses as disjunctions of literals and as sets of literals interchangeably. We will sometimes use a slightly abused notation $a(x, y) \subseteq a(w, x) \vee a(x, y)$ to denote that a set of literals of one clause is a subset of literals of another clause. The set of variables in a clause A is written as $vars(A)$ and the set of all terms by $terms(A)$. Terms can be variables or constants. A substitution θ is a mapping from variables of a clause A to terms

of a clause B. The next definition introduces the concepts of θ-subsumption and θ-equivalence [3].

Definition 1 (θ-subsumption). *Let A and B be clauses. The clause A θ-subsumes B (denoted by $A \preceq_\theta B$), 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 (written $A \approx_\theta B$).*

The notion of θ-subsumption was introduced by [3] as an incomplete approximation of implication. Let A and B be clauses. If $A \preceq_\theta B$ then $A \models B$ but the other direction of the implication does not hold in general. However, it does hold for non-self-resolving function-free clauses.

Example 1. Let us have clauses $A = a(X, Y) \vee a(Y, Z)$ and $B = a(c, d) \vee a(d, e) \vee a(f, d)$. Then $A \preceq_\theta B$ because, for $\theta = \{X/c, Y/d, Z/e\}$, we have $A\theta = a(c, d) \vee a(d, e) \subseteq B$.

Definition 2 (θ-Reduction). *Let A be a clause. If there is another clause R such that $A \approx_\theta R$ and $|R| < |A|$ then A is said to be θ-reducible. A minimal such R is called θ-reduction of A.*

Constraint satisfaction [4] with finite domains represents a class of problems closely related to the θ-subsumption problems and to relational-structure homomorphisms. In fact, as shown by [5], these problems are almost identical although the terminology differs.

Definition 3 (Constraint Satisfaction Problem). *A constraint satisfaction problem is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{V}$ is a set of variables, $\mathcal{D} = \{D_1, \ldots, D_{|\mathcal{V}|}\}$ is a set of domains of values (for each variable $v \in \mathcal{V}$), and $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ is a set of constraints. Every constraint is a pair (s, R), where s (*scope*) is an n-tuple of variables and R is an n-ary relation. An evaluation of variables θ satisfies a constraint $C_i = (s_i, R_i)$ if $s_i\theta \in R_i$. A solution is an evaluation that satisfies all constraints.*

The CSP representation of the problem of deciding $A \preceq_\theta B$ has the following form [6]. There is one CSP variable X_v for every variable $v \in vars(A)$. The domain of each of these CSP variables contains all terms from $terms(B)$. The set of constraints contains one k-ary constraint $C_l = (s_l, R_l)$ for each literal $l = pred_l(t_1, \ldots, t_k) \in A$. We denote by $I_{var} = (i_1, \ldots, i_m) \subseteq (1, \ldots, k)$ the indexes of variables in arguments of l (the other arguments might contain constants). The scope s_l of the constraint C_l is $(X_{t_{i_1}}, \ldots, X_{t_{i_m}})$ (i.e. the scope contains all CSP variables corresponding to variables in the arguments of literal l). The relation R_l of the constraint C_l is then constructed in three steps. First, a set L_l is created which contains all literals $l' \in B$ such that $l \preceq_\theta l'$ (note that checking θ-subsumption of two literals is a trivial linear-time operation). Then a relation R'_l is constructed from the arguments of these literals such that it contains a tuple $(t'_1, \ldots, t'_k)$ if and only if $l' = pred(t'_1, \ldots, t'_k) \in L_l$. Finally, the relation R_l of the constraint C_l is then the projection of R'_l on indexes I_{var} (only the elements of tuples which correspond to variables in l are retained).

Next, we exemplify this transformation process.

Example 2 (Converting θ-subsumption to CSP). Let us have clauses A and B as follows

$$A = hasCar(C) \vee hasLoad(C, L) \vee shape(L, box)$$
$$B = hasCar(c) \vee hasLoad(c, l_1) \vee hasLoad(c, l_2) \vee shape(l_2, box).$$

We now show how we can convert the problem of deciding $A \preceq_\theta B$ to a CSP problem. Let $\mathcal{V} = \{C, L\}$ be a set of CSP-variables and let $\mathcal{D} = \{D_C, D_L\}$ be a set of domains of variables from $\mathcal{V}$ such that $D_C = D_L = \{c, l_1, l_2\}$. Further, let $\mathcal{C} = \{C_{hasCar(C)}, C_{hasLoad(C,L)}, C_{shape(L,box)}\}$ be a set of constraints with scopes (C), (C, L) and (L) and with relations $\{(c)\}$, $\{(c, l_1), (c, l_2)\}$ and $\{(l_2)\}$, respectively. Then the constraint satisfaction problem given by $\mathcal{V}$, $\mathcal{D}$ and $\mathcal{C}$ represents the problem of deciding $A \preceq_\theta B$ as it admits a solution if and only if $A \preceq_\theta B$ holds.

The Gaifman (or primal) graph of a clause A is the graph with one vertex for each variable $v \in vars(A)$ and an edge for every pair of variables $u, v \in vars(A)$, $u \neq v$ such that u and v appear in a literal $l \in A$. Similarly, we define Gaifman graphs for CSPs. The Gaifman graph of a CSP problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ is the graph with one vertex for each variable $v \in \mathcal{V}$ and an edge for every pair of variables which appear in a scope of some constraint $c \in \mathcal{C}$. Gaifman graphs can be used to define treewidth of clauses or CSPs.

Definition 4 (Tree decomposition, Treewidth). *A tree decomposition of a graph $G = (V, E)$ is a labeled tree T such that*

- *Every node of T is labeled by a non-empty subset of V.*
- *For every edge $(v, w) \in E$, there is a node of T with label containing v, w.*
- *For every $v \in V$, the set of nodes of T with labels containing v is a connected subgraph of T.*

The width of a tree decomposition T is the maximum cardinality of a label in T minus 1. The treewidth of a graph G is the smallest number k such that G has a tree decomposition of width k. The treewidth of a clause is equal to the treewidth of its Gaifman graph. Analogically, the treewidth of a CSP is equal to the treewidth of its Gaifman graph.

It is easy to check that if a clause A has treewidth bounded by k then also the CSP representation of the problem of deciding $A \preceq_\theta B$ has treewidth bounded by k for any clause B. Constraint satisfaction problems with treewidth bounded by k can be solved in polynomial time by the k-consistency algorithm[1] [7]. If the k-consistency algorithm returns *false* for a CSP problem $\mathcal{P}$ then $\mathcal{P}$ is guaranteed to have no solutions. If it returns *true* then the problem may or may not have some solutions. Finally, if the k-consistency algorithm returns *true* and $\mathcal{P}$ has treewidth bounded by k then $\mathcal{P}$ is guaranteed to have a solution. It is known that due to the equivalence of CSPs and θ-subsumption, the problem of deciding θ-subsumption $A \preceq_\theta B$ can be solved in polynomial time when clause A has bounded treewidth.

[1] In this paper we follow the conventions of [2]. In other works, e.g. [7], what we call *k-consistency* is known as *strong $k+1$-consistency*.

Proposition 1. *We say that clause A is k-consistent w.r.t. clause B (denoted by $A \lhd_k B$) if and only if the k-consistency algorithm executed on the CSP representation of the problem of deciding $A \preceq_\theta B$ returns* true*. If A has treewidth at most k and $A \lhd_k B$ then $A \preceq_\theta B$.*

Proof. Follows directly from the solubility of CSPs with bounded treewidth by the k-consistency algorithm [2] and from the equivalence of CSPs and θ-subsumption shown earlier in this section.

4 Safe Reduction of Learning Examples

The learning task that we consider in this paper is fairly standard. We are given labelled learning examples encoded as first-order-logic clauses and we would like to find a classifier predicting the class labels of examples as precisely as possible. This task could be solved by numerous relational-learning systems. We aim at finding a reduction procedure that would allow us to reduce the number of literals in the examples while guaranteeing that the coverage of any hypothesis from a pre-fixed hypothesis language $\mathcal{L}$ would not be changed.

There are several settings for logic-based relational learning. We will work within the *learning from entailment setting* [8].

Definition 5 (Covering under Learning from Entailment). *Let $\mathcal{H}$ be a clausal theory and e be a clause. Then we say that $\mathcal{H}$ covers e under entailment if and only if $\mathcal{H} \models e$.*

The basic learning task is to find a clausal theory $\mathcal{H}$ that covers all positive examples and no negative examples and contains as few clauses as possible.

Definition 6 (Safe Equivalence and Safe Reduction under Entailment). *Let e and $\hat{e}$ be clauses and let $\mathcal{L}$ be a language specifying all possible hypotheses. Then $\hat{e}$ is said to be safely equivalent to e if and only if $\forall \mathcal{H} \in \mathcal{L} : (\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \hat{e})$. If e and $\widehat{e}$ are safely equivalent and $|\hat{e}| < |e|$ then $\widehat{e}$ is called safe reduction of e.*

Clearly, if we have a hypothesis $\mathcal{H} \in \mathcal{L}$ which splits the examples to two sets X and Y then this hypothesis $\mathcal{H}$ will also split the respective set of safely reduced examples to the sets $\widehat{X}$, $\widehat{Y}$ containing the safely reduced examples from the sets X and Y, respectively. Also, when predicting classes of test-set examples, any deterministic classifier that bases its decisions on the queries using the covering relation $\models$ will return the same classification even if we replace some of the examples by their safe reductions. The same is also true for propositionalization approaches that use the $\models$ relation to construct boolean vectors which are then processed by attribute-value-learners.

In this paper, we focus on hypothesis languages in the form of *non-resolving* clausal theories. Recall that we do not put any restrictions on the learning examples. The only restrictions are those put on hypotheses. A *non-resolving* clausal theory is a set of clauses such that no predicate symbol which appears in the head

of a clause appears also in the body of any clause. The main reason why we start with non-resolving clausal theories is that logical entailment $\mathcal{H} \models A$, for a non-resolving clausal theory $\mathcal{H}$ and a clause A, can be checked using θ-subsumption. If there is a clause $H \in \mathcal{H}$ such that $H \preceq_\theta A$ then $\mathcal{H} \models A$, otherwise $\mathcal{H} \not\models A$. The non-resolving hypothesis languages as we defined them here are also sometimes called *pattern languages*.

We start by defining x-subsumption and x-equivalence which are weaker versions of θ-subsumption and θ-equivalence. The notions of x-subsumption and x-equivalence will be central tools used in this section.

Definition 7 (x-subsumption, x-equivalence). *Let X be a possibly infinite set of clauses. Let A, B be clauses not necessarily from X. We say that A x-subsumes B w.r.t. X (denoted by $A \preceq_x B$) if and only if $(C \preceq_\theta A) \Rightarrow (C \preceq_\theta B)$ for every clause $C \in X$. If $A \preceq_x B$ and $B \preceq_x A$ then A and B are called x-equivalent w.r.t. X (denoted by $A \approx_x B$). For a given set X, the relation $\preceq_x$ is called x-subsumption on X and the relation $\approx_x$ is called x-equivalence on X.*

For example, the set X can consist of clauses having treewidth bounded by k or having hypertreewidth bounded by l or having at most m variables etc. When it is clear from the context, we omit the phrase *w.r.t. X* from *A x-subsumes B w.r.t. X*. The x-equivalence w.r.t. the set X is closely related to safe equivalence w.r.t. a set $\mathcal{L} \subseteq 2^X$ containing only non-resolving clausal theories composed of clauses from X. If two learning examples are x-equivalent w.r.t. the set of clauses X then they are also safely equivalent w.r.t. the set of clausal theories $\mathcal{L}$.

Example 3. Let us have the following two clauses: $C = e(A,B) \vee e(B,C) \vee e(C,A)$ and $D = e(A,B) \vee e(B,C) \vee e(C,D) \vee e(D,A)$. For these clauses, it holds $C \preceq_x D$ and $D \preceq_x C$ w.r.t. the set of clauses with treewidth at most 1. On the other hand, $C \not\preceq_x D$, $D \not\preceq_x C$ for sets of clauses with treewidth at most k where $k > 1$. This is because the treewidth of C and D is 2.

The next proposition states basic properties of x-subsumption and x-equivalence (the proof is simple but is omitted due to lack of space).

Proposition 2. *Let X be a set of clauses. Then x-subsumption w.r.t. X is a transitive and reflexive relation on clauses and x-equivalence w.r.t. X is an equivalence relation on clauses.*

Definition 7 provides no efficient way to decide x-subsumption between two clauses as it demands θ-subsumption of an infinite number of clauses to be tested in some cases. The next proposition provides a necessary condition for x-subsumption. It will be the basic tool that we exploit in this section to develop methods for safely reducing learning examples.

Proposition 3. *Let X be a set of clauses. If $\preceq_x$ is x-subsumption on X and $\lhd_x$ is a relation such that:*

1. *If $A \lhd_x B$ and $C \subseteq A$ then $C \lhd_x B$.*

2. If $A \in X$, ϑ is a substitution and $A\vartheta \lhd_x B$ then $A \preceq_x B$.

Then $(A \lhd_x B) \Rightarrow (A \preceq_x B)$ for any two clauses A, B (not necessarily from X).

Proof. We need to show that if $A \lhd_x B$ then $(C \preceq_\theta A) \Rightarrow (C \preceq_\theta B)$ for all clauses $C \in X$. First, if $A \lhd_x B$ and $C \not\preceq_\theta A$ then the proposition holds trivially. Second, $C \preceq_\theta A$ means that there is a substitution ϑ such that $C\vartheta \subseteq A$. This implies $C\vartheta \lhd_x B$ using the condition 1. Now, we can use the second condition which gives us $C \preceq_x B$ (note that $C \in X$ and $C\vartheta \lhd_x B$). Finally, we get $C \preceq_\theta B$ using Definition 7 because $C \in X$.

Proposition 3 can be used to check if two learning examples e and $\widehat{e}$ are equivalent w.r.t. hypotheses from a fixed hypothesis language. It can be therefore used to search for safe reductions of learning examples. This is formalized in the next proposition. Note that this proposition does not say that e and $\widehat{e}$ are equivalent. It merely says that they are equivalent when being used as learning examples in the *learning from entailment* setting with hypotheses drawn from a fixed set.

Proposition 4. *Let $\mathcal{L}$ be a hypothesis language containing only non-resolving clausal theories composed of clauses from a set X and let $\lhd_X$ be a relation satisfying conditions 1 and 2 from Proposition 3 on the set X. If e and $\widehat{e}$ are learning examples (not necessarily from X), $e \lhd_X \widehat{e}$ and $\widehat{e} \lhd_X e$ then for any $\mathcal{H} \in \mathcal{L}$ it holds $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \widehat{e})$. Moreover, if $|\widehat{e}| < |e|$ then $\widehat{e}$ is a safe reduction of e under entailment.*

Proof. First, $e \lhd_X \widehat{e}$ and $\widehat{e} \lhd_X e$ imply $e \approx_X \widehat{e}$ (where $\approx_X$ denotes x-equivalence on the set X). Then for any non-resolving clausal theory $\mathcal{H} \in \mathcal{L}$ we have $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \widehat{e})$ because for any clause $A \in X$ we have $(A \preceq_\theta e) \Leftrightarrow (A \preceq_\theta \widehat{e})$ (from $e \approx_X \widehat{e}$). This together with $|\widehat{e}| < |e|$ means that $\widehat{e}$ is a safe reduction of e under entailment w.r.t. hypothesis language $\mathcal{L}$.

We will use Propositions 3 and 4 for showing that certain procedures which transform learning examples always produce safe reductions of these examples. Specifically, we will use them to show that k-consistency algorithm can be used for computing safe reductions of learning examples w.r.t. hypothesis sets composed of clauses with bounded treewidth.

We start with two simpler transformation methods for which Propositions 3 and 4 are not actually needed. For the first transformation method, we assume to have a fixed hypothesis language $\mathcal{L}_\mathcal{U}$ consisting of non-resolving clausal theories which contain only constants from a given set $\mathcal{U}$. The transformation then gets a clause A on its input and produces a new clause $\tilde{A}$ by *variabilizing* constants in A which are not contained in $\mathcal{U}$. It is easy to check that for any such A and $\tilde{A}$ it must hold $A \approx_x \tilde{A}$ w.r.t. the set of clauses containing only constants from $\mathcal{U}$. Therefore A and $\tilde{A}$ are safely equivalent w.r.t. $\mathcal{L}$. We can think of the constants not used in a hypothesis language $\mathcal{L}$ as identifiers of objects whose exact identity is not interesting for us. Such constants can appear e.g. when we describe molecules and we want to give names to atoms in the molecules with no actual meaning.

Another simple transformation which produces safely equivalent clauses is based on θ-reduction. In this case the set of clauses X can be arbitrary. The transformation gets a clause A on its input and returns its θ-reduction. The x-equivalence of the clause A and its θ-reduction follows from the fact that θ-subsumption is an x-subsumption w.r.t. the set of all clauses.

Importantly, transformations which produce x-equivalent clauses w.r.t. a set X can be chained due to transitivity of x-subsumption. So, for example, if we have a hypothesis language $\mathcal{L}_{\mathcal{U}}$ consisting of non-resolving clausal theories which contain only constants from a pre-fixed set $\mathcal{U}$ and we want to safely reduce a clause A then we can first variabilize it and then reduce it using θ-reduction.

Example 4. Let us have an example

$$e = edge(a, b, 1) \vee edge(b, a, 2) \vee edge(b, c, 2) \vee edge(c, d, 1) \vee edge(d, a, 2)$$

and a hypothesis language $\mathcal{L}$ containing arbitrary non-resolving clausal theories with the set of allowed constants $\mathcal{U} = \{1, 2\}$. We variabilize e and obtain clause

$$\tilde{e} = edge(A, B, 1) \vee edge(B, A, 2) \vee edge(B, C, 2) \vee edge(C, D, 1) \vee edge(D, A, 2).$$

Now, e and $\tilde{e}$ are safely equivalent w.r.t. to hypotheses from $\mathcal{L}$. Next, we obtain a safe reduction of e by computing θ-reduction of $\tilde{e}$ which is $\widehat{e} = edge(A, B, 1)$ $\vee$ $edge(B, A, 2)$.

Next, we describe a transformation method which assumes the hypothesis languages to consist only of clauses with bounded treewidth. Unlike the exponential-time method based on θ-reduction, this method runs in polynomial time. Interestingly, it does not need any restrictions (e.g. bounded treewidth) on the learning examples which are reduced. The reduction method is based on x-subsumption on the set X_k of clauses with treewidth at most k. We start by showing that x-subsumption w.r.t. X_k can be checked using the k-consistency algorithm. We do this by showing that the k-consistency relation $\lhd_k$ on clauses satisfies the conditions from Proposition 3.

Proposition 5. *Let X_k be a set containing only clauses with treewidth at most k. For any two clauses A, B, if $A \lhd_k B$ (i.e. if A is k-consistent w.r.t. B) then $A \preceq_x B$ w.r.t. the set X_k.*

Proof. We will show that the conditions of Proposition 3 are satisfied by $\lhd_k$ from which the validity of the proposition will follow. First: *If $C \subseteq A$ and $A \lhd_k B$ then $C \lhd_k B$.* which is obviously true. Second: *If C is a clause with treewidth bounded by k and $C\theta \lhd_k D$ then $C \preceq_\theta D$.* Checking $C\theta \lhd_k D$ is equivalent to checking k-consistency of the original CSP representation of $C \preceq_\theta D$ problem where we added additional constraints to enforce consistency with the substitution θ. If this *restricted* problem is still k-consistent then also for the original problem it must have held $C \lhd_k D$ and consequently $C \preceq_\theta D$ because C has treewidth at most k (using Proposition 1).

The safe reduction method based on k-consistency works as follows. We suppose that there is a set $\mathcal{U}$ of constants which are allowed in the hypothesis language $\mathcal{L}_k$ and that the hypotheses in $\mathcal{L}_k$ consist only of clauses with treewidth at most k. The method gets a clause A and variabilizes all constants not contained in $\mathcal{U}$. The result is a clause $\tilde{A}$ which is also safely equivalent to A w.r.t. the hypothesis language $\mathcal{L}_k$. This clause is then reduced by so-called *literal-elimination algorithm* (described in proof of Proposition 6) which is based on the k-consistency algorithm, always produces a clause $\widehat{A}$ which is safely equivalent to A w.r.t. $\mathcal{L}_k$ and contains at most as many literals as a safe reduction obtained by the method based on θ-reduction as the next proposition shows. Moreover, it runs in polynomial time.

Proposition 6. *Let $\mathcal{L}_k$ be a set of non-resolving hypotheses containing only clauses with treewidth at most k. Let C be a clause and $\widehat{C}_\theta$ its θ-reduction. We can find a clause $\widehat{C}_k$ such that $C \approx_x \widehat{C}_k$ w.r.t. to $\mathcal{L}_k$ and $|\widehat{C}_k| \leq |\widehat{C}_\theta|$ in time $\mathcal{O}\left(|C|^{2k+3}\right)$ by the "literal-elimination algorithm".*

Proof. The *literal k-elimination algorithm* works as follows. First, it sets $C' = C$ and then performs the following process iteratively for each literal $l \in C$: It checks if $C \lhd_k C' \setminus \{l\}$ and if so then it sets $C' \leftarrow C' \setminus \{l\}$. First, it follows from transitivity of k-equivalence that $\widehat{C}_k \approx_k C$. What remains to be shown is that the resulting clause $\widehat{C}_k$ will not be bigger than $\widehat{C}_\theta$. Let us assume, for contradiction, that $|\widehat{C}_k| > |\widehat{C}_\theta|$. Since $\widehat{C}_k \subseteq C$, it must hold $\widehat{C}_k \preceq_\theta \widehat{C}_\theta$ which means that $\widehat{C}_k$ must be θ-reducible. When $\widehat{C}_k$ is θ-reducible, there must be a literal $l \in \widehat{C}_k$ such that $\widehat{C}_k \preceq_\theta \widehat{C}_k \setminus \{l\}$. θ-subsumption implies k-consistency (for clauses of arbitrary treewidth) therefore it also holds $\widehat{C}_k \lhd_k \widehat{C}_k \setminus \{l\}$. However, then l should have been removed by the literal-elimination algorithm which is a contradiction with $\widehat{C}_k$ being output of it. As for the running time of the algorithm, k-consistency can be checked in time $\mathcal{O}(|C|^{2k+2})$ [2] and it is invoked exactly $|C|$ times by the above procedure which gives us the runtime $\mathcal{O}(|C|^{2k+3})$.

We can find even smaller safely equivalent clauses w.r.t. $\mathcal{L}_k$ for a clause C by a *literal-substitution algorithm* with just a slightly higher runtime $\mathcal{O}(|C|^{2k+4})$. This algorithm first runs the *literal-elimination algorithm* and then tries to further reduce its output C' as follows: For each pair of literals $l, l' \in C'$ it constructs a substitution $\theta : vars(l) \to vars(l')$ and checks if $C'\theta \lhd_k C'$ and if so, it sets $C' \leftarrow C'\theta$. It is easy to check that it always holds $C \approx_x C'$ w.r.t. the set of clauses with treewidth at most k. The algorithm runs in time $\mathcal{O}(|C|^{2k+4})$ as it performs $\mathcal{O}(|C|^2)$ k-consistency checks.

The clauses with bounded treewidth are not the only ones for which efficient safe reduction can be derived. For example, it is possible to derive a completely analogical safe reduction w.r.t. acyclic clauses, which can have arbitrary high treewidth but despite that admit a polynomial-time θ-subsumption checking algorithm. The only difference would be the use of generalized arc-consistency algorithm [7] instead of the k-consistency test.

5 Experimental Evaluation of Safe Reduction

We experimentally evaluate usefulness of the *safe reduction of learning examples* with real-world datasets and a state-of-the-art relational learning system nFOIL [9]. We implemented *literal-elimination* and *literal-substitution* algorithms for treewidth 1, i.e. for tree-like clausal theories. We used the efficient algorithm AC-3 [10] for checking 1-consistency[2]. We forced nFOIL to construct only clauses with treewidth 1 using its *mode declaration* mechanism. We used three datasets in the experiments: *predictive toxicology challenge* [11], *CAD* [12] and *hexose-binding proteins* [13]. The PTC dataset contains descriptions of 344 molecules classified according to their toxicity for male rats. The molecules are described using only *atom* and *bond* information. The CAD dataset contains descriptions of 96 class-labelled product-structure designs. Finally, the hexose-binding dataset contains 80 hexose-binding and 80 non-hexose-binding protein domains. Following [13] we represent the protein domains by atom-types and atom-names (each atom in an amino acid has a unique name) and pair-wise distances between the atoms which are closer to each other than some threshold value. We performed two experiments with the last mentioned dataset for cut-off set to 1 Angstrom and 2 Angstroms with which we were able to achieve accuracies as high as [13] have achieved using Aleph.

We applied the *literal-elimination* algorithm followed by *literal-substitution* algorithm on the three datasets. The compression rates (i.e. ratios of number of literals in the reduced learning examples divided by the number of literals in the original non-reduced examples) are shown in the left panel of Figure 2. The right panel of Figure 2 then shows the time needed to run the reduction algorithms on the respective datasets. We note that these times are generally negligible compared to runtimes of nFOIL.

Next, we used nFOIL to learn predictive models and evaluated them using 10-fold cross-validation. For all experiments with the exception of the hexose-binding dataset with cut-off value 2 Angstroms, where we used beam-size 50, we used beam-size 100. From one point of view, this is much higher than the beam-sizes used by [9], but on the other hand, we have the experience that this allows nFOIL to find theories which involve longer clauses and at the same time have higher predictive accuracies. The runtimes of nFOIL operating on reduced and non-reduced data are shown in the left panel of Figure 3. It can be seen that the reduction was beneficial in all cases but that the most significant speed-up of more than an order of magnitude was achieved on Hexose data. This could be attributed to the fact that nFOIL constructed long clauses on this dataset and the covering test used by it had not probably been optimized. So, in principle, nFOIL could be made faster by optimizing the efficiency of its covering test. The main point, however, is that we can speed-up the learning process for almost any relational learning algorithm merely by preprocessing its input.

[2] Note again the terminology used in this paper following [2]. In CSP-literature, it is often common to call 2-consistency what we call 1-consistency.

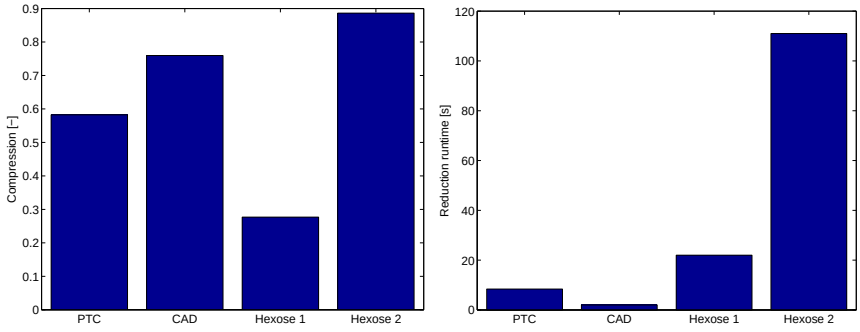

Fig. 2. Left: Compression rates achieved by *literal-substitution algorithm* on four datasets (for treewidth 1). **Right:** Time for computing reductions of learning examples on four datasets (for treewidth 1).

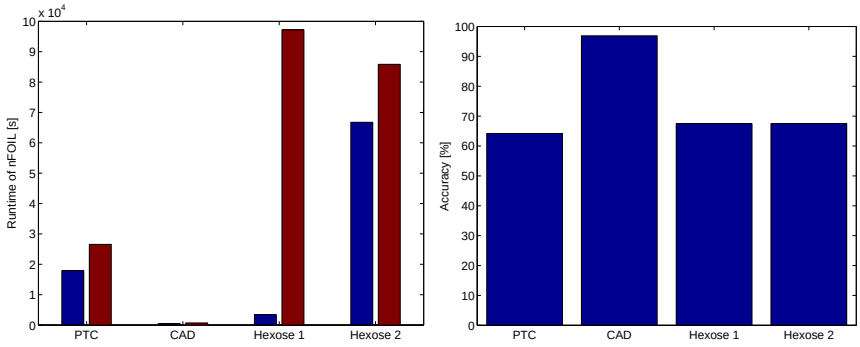

Fig. 3. Left: Runtime of nFOIL on reduced (blue) and non-reduced (red) datasets. **Right:** Predictive accuracies of nFOIL on four datasets estimated by 10-fold cross-validation.

6 Conclusions

We have introduced a novel concept called *safe reduction*. We have shown how it can be used to safely reduce learning examples (without affecting learnability) which makes it possible to speed-up many relational learning systems by merely preprocessing their input. The methods that we have introduced run in polynomial time for hypothesis languages composed of bounded-treewidth or acyclic clauses and can reduce the learning examples always at least as much as a conventional exponential-time method.

Acknowledgements

This work was supported by the Czech Grant Agency through project 103/11/2170 *Transferring ILP techniques to SRL* and by the Czech Technical University in Prague through the student grant competition project SGS11/155/OHK3/3T/13.

Appendix: The k-Consistency Algorithm

In this section, we briefly describe the k-consistency algorithm. The description is based on the presentation by Atserias et al. [2]. Let us have a CSP $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V}$ is the set of variables, $\mathcal{D}$ is the set of domains of the variables and $\mathcal{C}$ is the set of constraints. A partial solution ϑ is an evaluation of variables from $\mathcal{V}' \subseteq \mathcal{V}$ which is a solution of the sub-problem $\mathcal{P}' = (\mathcal{V}', \mathcal{D}, \mathcal{C})$. If ϑ and φ are partial solutions, we say that φ extends ϑ (denoted by $\vartheta \subseteq \varphi$) if $Supp(\vartheta) \subseteq Supp(\varphi)$ and $V\vartheta = V\varphi$ for all $V \in Supp(\vartheta)$, where $Supp(\vartheta)$ and $Supp(\varphi)$ denote the sets of variables which are affected by the respective evaluations ϑ and φ.

The k-consistency algorithm then works as follows:

1. Given a constraint satisfaction problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ and a positive integer k.
2. Let H be the collection of all partial solutions ϑ with $|Supp(\vartheta)| < k + 1$.
3. For every $\vartheta \in H$ with $|Supp(\vartheta)| \leq k$ and every $V \in \mathcal{V}$, if there is no $\varphi \in H$ such that $\vartheta \subseteq \varphi$ and $V \in Supp(\varphi)$, remove ϑ and all its extensions from H.
4. Repeat step 3 until H is unchanged.
5. If H is empty return *false*, else return *true*.

References

1. Kuželka, O., Železný, F.: Seeing the world through homomorphism: An experimental study on reducibility of examples. In: ILP. (2011) 138–145
2. Atserias, A., Bulatov, A., Dalmau, V.: On the power of k-consistency. In: Proceedings of ICALP-2007. (2007) 266–271
3. Plotkin, G.: A note on inductive generalization. Edinburgh University Press (1970)
4. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
5. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. SIAM J. Comput. **28**(1) (1998) 57–104
6. Maloberti, J., Sebag, M.: Fast theta-subsumption with constraint satisfaction algorithms. Machine Learning **55**(2) (2004) 137–174
7. Rossi, F., van Beek, P., Walsh, T., eds.: Handbook of Constraint Programming. Elsevier (2006)
8. De Raedt, L.: Logical settings for concept-learning. Artif. Intell. **95**(1) (1997) 187–201
9. Landwehr, N., Kersting, K., Raedt, L.D.: Integrating naïve bayes and FOIL. Journal of Machine Learning Research **8** (2007) 481–507
10. Mackworth, A.: Consistency in networks of relations. Artificial Intelligence **8**(1) (1977) 99–118
11. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000-2001. Bioinformatics **17**(1) (2001) 107–108
12. Žáková, M., Železný, F., Garcia-Sedano, J., Tissot, C.M., Lavrač, N., Křemen, P., Molina, J.: Relational data mining applied to virtual engineering of product designs. In: ILP06. Volume 4455 of LNAI., Springer (2007) 439–453
13. Nassif, H., Al-Ali, H., Khuri, S., Keirouz, W., Page, D.: An Inductive Logic Programming approach to validate hexose biochemical knowledge. In: Proceedings of the 19th International Conference on ILP, Leuven, Belgium (2009) 149–165