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. Feature selection methods often improve the performance of attribute-value learning. We explore whether also in relational learning, examples in the form of clauses can be reduced in size to speed up learning *without affecting the learned hypothesis*. To this end, we introduce the notion of safe reduction: a safely reduced example cannot be distinguished from the original example *under the given hypothesis language bias*. Next, we consider the particular, rather permissive bias of bounded treewidth clauses. We show that under this hypothesis bias, examples of arbitrary treewidth can be reduced efficiently. The bounded treewidth bias can be replaced by other assumptions such as acyclicity with similar benefits. We evaluate our approach on four data sets with the popular system Aleph and the state-of-the-art relational learner nFOIL. On all four data sets we make learning faster for nFOIL, achieving an order-of-magnitude speed up on one of the data sets, and more accurate for Aleph.

1 Introduction

Reducing the complexity of input data is often beneficial for learning. In attribute-value learning, a wide range of *feature selection* methods is available [1]. These methods try to select a strict subset of the original example features (attributes) while maintaining or even improving the performance of the model learned from it with respect to that learned from the original feature set. For binary classification tasks with Boolean features, the REDUCE [2] algorithm has been proposed that removes so called *irrelevant* features. For any model learned with the original feature set, a model with same or better fit on the learning examples may be expressed without the irrelevant features. In the later work [3], the REFER algorithm extended REDUCE to the multiple-class learning setting.

In inductive logic programming—an important framework for relational learning [4]—examples are not expressed as tuples of feature values but rather take the form of logical constructs such as first-order clauses. Feature-selection methods are thus not applicable to simplify such learning examples. Here we are interested to see whether also first-order clausal examples can somehow be reduced while guaranteeing that the set of logical formulas which can be induced from such reductions would not be affected.

A. Appice et al. (Eds.): NFMCP 2012 Workshop, LNAI 7765, pp. 17–32, 2013.

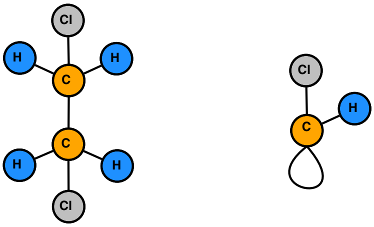


Fig. 1. A learning example and its reduction

An obvious approach would be to look for θ-reductions [5] of the input clauses. A θ-reduction of a clause is a smaller, but subsumption-equivalent (and thus also logically equivalent) clause. We have explored an approach based on θ-subsumption before in [6], achieving learning speed-up factors up to 2.63. However, the main problem of θ-reduction is that finding it is an NP-hard problem, rendering the approach practically unfeasible in domains with large examples such as those describing protein structures [7].

Here we follow the key idea that the complexity curse can be avoided by sacrificing part of the generality of θ-reduction. In particular, we will look for reductions which may not be equivalent to the original example in the logical sense, but which are equivalent *given the language bias* of the learning algorithm. In other words, if the learning algorithm is not able to produce a hypothesis covering the original example but not covering its reduction (or vice versa), the latter two may be deemed equivalent. For instance, consider the clausal example $\leftarrow \mathsf{atom}(\mathsf{a1}) \wedge \mathsf{carbon}(\mathsf{a1}) \wedge \mathsf{bond}(\mathsf{a1}, \mathsf{a2}) \wedge \ldots$, whose entire structure is shown in the left of Figure 1. Assume that all terms in hypotheses are variables and hypotheses must have *treewidth* at most 1 or be *acyclic*. Then the learning example is equivalent to the simpler one shown in the right of the figure.

Our first main contribution is a formal framework for example reduction based on the given language bias for hypotheses. In this framework, we prove two propositions which can be used for showing that certain procedures which transform learning examples always produce reductions equivalent to the original examples under the given bias.

Our second main contribution is the application of the above framework to the specific bias of *bounded treewidth* clauses. We show that in this case, interestingly, learning examples can be reduced in polynomial time, and moreover, that, in some cases, they can be reduced even more than they would be using the NP-hard θ-reduction.

Intuitively, a clause viewed as a graph (not necessarily a tree) has a small treewidth if it can be recursively decomposed into small subgraphs that have small overlap [8]. As the previous paragraph indicates, the benefits gained from the bounded treewidth assumption are significant. Notably though, the price we pay for them is not too high in that the bias would be over-restrictive. First remind that, as a hypothesis bias, it only constrains the learned clauses, and not the learning examples. Second, low treewidth is in fact characteristic of clauses induced in typical ILP experiments. In [9] we observed that all clauses learned by the ILP system Progol in all the conducted experiments had treewidth 1 although this had not been stipulated by the language bias. Similarly, in experiments in another study [10], all clauses learned by the systems nFOIL and kFOIL were of treewidth 1 after the removal of the variable formally identifying the learning example. These observations become plausible when viewing the exemplary chemical-domain clauses shown in the leftmost column of Table 1, which have the respective treewidths 1 and 2.

A salient feature of our approach is its general application scope. Indeed, example reduction can take place independently of the type of ILP learner employed subsequently. While we evaluate the approach in the standard ILP setting of learning from entailment, it is also relevant to propositionalization [11], which aims at the construction of feature-based descriptions of relational examples. Interestingly, propositionalization can simultaneously benefit from both the relational example reduction step employed before the construction of features, and any feature selection algorithm applied subsequently on the constructed feature set. In this sense, our approach is complementary to standard feature selection methods.

The rest of the paper is structured as follows. In the next section we review the preliminaries for the study, namely θ-subsumption and reduction, their correspondence to the constraint satisfaction problem model, and the concepts of tree decomposition and treewidth. Section 3 presents the framework for safe reduction of relational examples under a given hypothesis language bias. Section 4 instantiates the framework to the langauge bias of bounded-treewidth clauses and shows that reduction under this bias can be conducted effectively. We experimentally evaluate our method in Section 5 and conclude in Section 6.

2 Preliminaries: Logic, Constraint Satisfaction, Treewidth

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 [5].

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 [5] 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 [12] 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 [13], 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 [14]. 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 $l = \mathsf{pred}_l(t_1, \ldots, t_k) \in A$. 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 $l' = \mathsf{pred}(t'_1, \ldots, t'_k) \in L_l$. 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

$$A = \mathsf{hasCar}(C) \vee \mathsf{hasLoad}(C, L) \vee \mathsf{shape}(L, \mathsf{box})$$

$$B = \mathsf{hasCar}(c) \vee \mathsf{hasLoad}(c, l_1) \vee \mathsf{hasLoad}(c, l_2) \vee \mathsf{shape}(l_2, \mathsf{box}).$$

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 $\mathcal{C} = \{C_{\mathsf{hasCar}(C)}, C_{\mathsf{hasLoad}(C,L)}, C_{\mathsf{shape}(L,\mathsf{box})}\}$ 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. Likewise, the treewidth of a CSP is equal to the treewidth of its Gaifman graph.

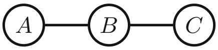
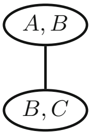
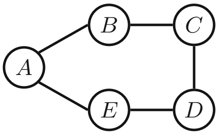
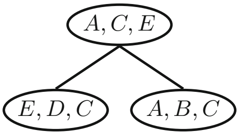
An illustration of Gaifman graphs of two exemplar clauses and their tree-decompositions is shown in Table 1. Note that tree decomposition is not unique.

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] [16]. 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.

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 $A \preceq_\theta B$.*

[1] In this paper we follow the conventions of [15]. In other works, e.g. [16], what we call *k-consistency* is known as *strong $k + 1$-consistency*.

Table 1. An illustration of Gaifman graphs and tree-decompositions of clauses

Clause	Gaifman graph	Tree decomposition
$\leftarrow \mathsf{atm}(A,\mathsf{h})\wedge$ $\mathsf{bond}(A,B,1)\wedge\mathsf{atm}(B,\mathsf{c})\wedge$ $\mathsf{bond}(B,C,2)\wedge\mathsf{atm}(C,\mathsf{o})$		
$\leftarrow \mathsf{bond}(A,B,1)\wedge$ $\mathsf{bond}(B,C,1)\wedge\mathsf{bond}(C,D,1)\wedge$ $\mathsf{bond}(D,E,1)\wedge\mathsf{bond}(E,A,1)$		

Proof. Follows directly from the solubility of CSPs with bounded treewidth by the k-consistency algorithm [15] and from the equivalence of CSPs and θ-subsumption shown earlier in this section.

3 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* [17].

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 two 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 $\mathcal{H} \not\models A$.

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 [18] 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.

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.*

Proof. These properties of x-subsumption and x-equivalence can be shown very easily.

1. Transitivity of x-subsumption: Let $A \preceq_X B$ and $B \preceq_X C$. We need to show that then necessarily also $A \preceq_X C$, i.e. that for any clause $D \in X$ such that $D \preceq_\theta A$ it also holds $D \preceq_\theta C$. This is straightforward because if $D \preceq_\theta A$ then $D \preceq_\theta B$ (from $A \preceq_X B$) and also $D \preceq_\theta C$ (from $B \preceq_X C$).
2. Reflexivity of x-subsumption: obvious.
3. x-equivalence is an equivalence relation: Reflexivity and transitivity of x-equivalence follow from reflexivity and transitivity of x-subsumption. It remains to show that x-equivalence is also symmetric but that follows immediately from $(A \approx_X B) \Leftrightarrow (A \preceq_X B \wedge B \preceq_X A)$.

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

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

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

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

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 $\widehat{e} = \mathsf{edge}(A, B, 1) \vee \mathsf{edge}(B, A, 2)$.

4 Reduction under the Bounded Treewidth Assumption

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 time polynomial in the size of the reduced clause (though, with a multiplicative factor exponential in the fixed maximum treewidth of allowed hypotheses). 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).

We leave the question open whether x-subsumption w.r.t. the set of clauses with treewidth at most k also implies k-consistency.

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* which is based on the k-consistency algorithm, always produces a clause $\widehat{A}$ which is safely equivalent to A w.r.t. $\mathcal{L}_k$ as Proposition 6 shows. Moreover, it runs in time polynomial in the size of the reduced clause (though, with a multiplicative factor exponential in the fixed maximum treewidth of allowed hypotheses).

Literal-elimination algorithm:

1. Given a clause A for which the x-reduction should be computed.
2. Set $A' := A$, $CheckedLiterals := \{\}$.
3. Select a literal L from $A' \setminus CheckedLiterals$. If there is no such literal, return A' and finish.

4. If $A \lhd_k A' \setminus \{L\}$ then set $A' := A' \setminus \{L\}$, else add L to *CheckedLiterals*.
5. Go to step 3.

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 $\widehat{C}_\theta$ be the maximal θ-reduction of a subset of the literals in C. 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. 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 θ-reducible. Let us assume, for contradiction, that $|\widehat{C}_k|$ is 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 $\mathcal{O}(|C|^{2k+2})$ [15] 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 $\theta : vars(l) \rightarrow vars(l')$ 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 [16] 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 two relational learning systems – the popular system Aleph and the state-of-the-art system nFOIL [19]. 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 [20] for checking 1-consistency[2]. We forced nFOIL and Aleph to construct only clauses with

[2] Note again the terminology used in this paper following [15]. In CSP-literature, it is often common to call 2-consistency what we call 1-consistency.

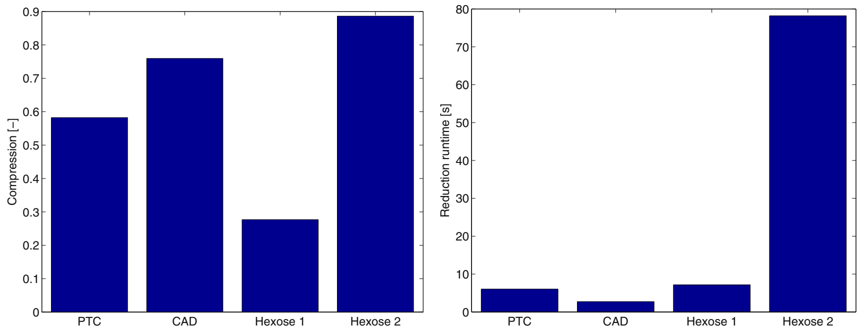


Fig. 2. Left: Compression rates achieved by *literal-substitution algorithm* on four datasets (for treewidth 1). **Right:** Time for computing reductions of learning examples on four datasets (for treewidth 1).

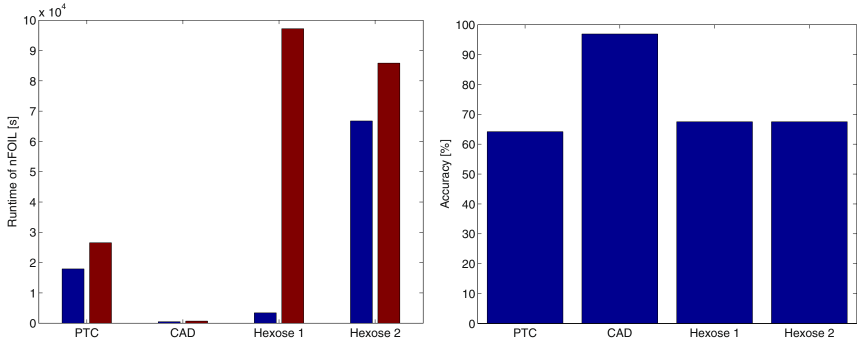


Fig. 3. Left: Runtime of nFOIL on reduced (blue) and non-reduced (red) datasets. **Right:** Predictive accuracies of nFOIL on four datasets estimated by 10-fold cross-validation.

treewidth 1 using their *mode declaration* mechanisms. We used three datasets in the experiments: *predictive toxicology challenge* [21], *CAD* [22] and *hexose-binding proteins* [7]. 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 [7] 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.

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 and with the exception of Hexose ver. 2 also to runtimes of Aleph.

5.1 Experiments with nFOIL

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 [19], 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. The right panel of Figure 3 shows nFOIL's predictive accuracies (estimated by 10-fold cross-validation). The accuracies were not affected by the reductions. The reason is that (unlike Aleph) nFOIL exploits learning examples only through the entailment queries.

5.2 Experiments with Aleph

We performed another set of experiments using the relational learning system Aleph. Aleph restricts its search space by bottom-clauses. After constructing a bottom-clause it searches for hypotheses by enumerating subsets of literals of the bottom-clause. When we reduce learning examples, which also means reduction of bottom-clauses, we are effectively reducing the size of Aleph's search space. This means that Aleph can construct longer clauses earlier than if it used non-reduced examples. On the other hand, this also implies that, with the same settings, Aleph may run longer on reduced data than on non-reduced data. That is because computing coverage of longer hypotheses is more time-consuming. Theories involving longer clauses may often lead to more accurate predictions. For these reasons, we measured not only runtime and accuracy, but also the average number of learnt rules and the average number of literals in these rules on reduced and non-reduced data.

We ran Aleph on reduced and non-reduced versions of the datasets and evaluated it using 10-fold cross-validation. We used the literal elimination algorithm

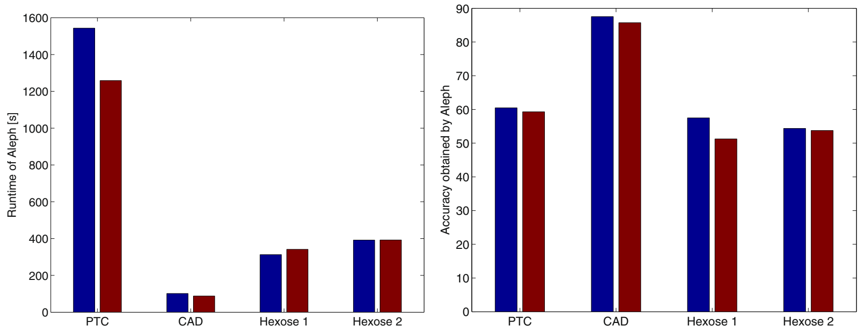


Fig. 4. Left: Runtime of Aleph on reduced (blue) and non-reduced (red) datasets. **Right:** Predictive accuracies of Aleph on reduced (blue) and non-reduced (red) datasets estimated by 10-fold cross-validation.

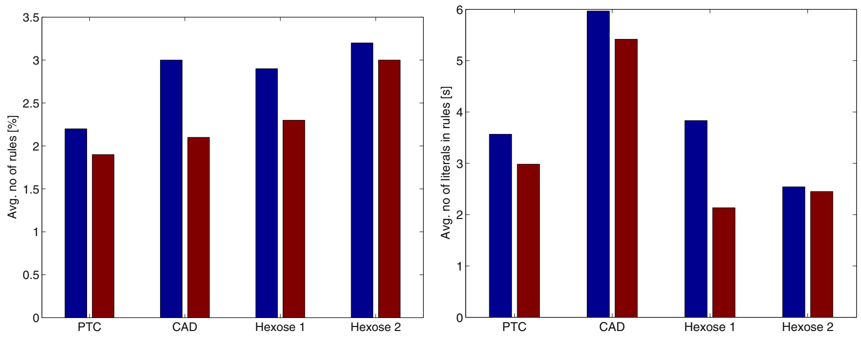


Fig. 5. Left: Average number of rules generated by Aleph on reduced (blue) and non-reduced (red) datasets. **Right:** Average number of literals in rules generated by Aleph on reduced (blue) and non-reduced (red) datasets.

for reducing examples. We set the maximum number of explored *nodes* to 50000, the *noise* parameter to 1% of the number of examples in the respective datasets. The runtime in the performed experiments was higher for reduced versions of datasets PTC and CAD, the same for Hexose 2 and lower for Hexose 1 than for their non-reduced counterparts (see left panel of Figure 4). The accuracies were higher for reduced versions of all four datasets (see right panel of Figure 4). Similarly, the average number of rules, as well as the average number of literals in the rules, was higher for the reduced versions of all four datasets (see Figure 5). These results confirm the expectation that Aleph should be able to construct longer hypotheses on reduced datasets which, in turn, should result in higher predictive accuracies.

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 clauses with treewidth bounded by a fixed constant. The bounded-treewidth assumption, while arguably appropriate in ILP, can be replaced by other kinds of assumptions such as acyclicity.

Acknowledgements. This work was supported by the Czech Grant Agency through project 103/11/2170 *Transferring ILP techniques to SRL.*

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. [15]. 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. Liu, H., Motoda, H., Setiono, R., Zhao, Z.: Feature selection: An ever evolving frontier in data mining. Journal of Machine Learning Research - Proceedings Track 10, 4–13 (2010)
2. Lavrac, N., Gamberger, D., Jovanoski, V.: A study of relevance for learning in deductive databases. J. Log. Program. 40(2-3), 215–249 (1999)
3. Appice, A., Ceci, M., Rawles, S., Flach, P.A.: Redundant feature elimination for multi-class problems. In: ICML, vol. 69 (2004)
4. Raedt, L.D.: Logical and Relational Learning: From ILP to MRDM (Cognitive Technologies). Springer-Verlag New York, Inc. (2008)
5. Plotkin, G.D.: A note on inductive generalization. Machine Intelligence 5, 153–163 (1970)

6. Kuželka, O., Železný, F.: Seeing the world through homomorphism: An experimental study on reducibility of examples. In: Frasconi, P., Lisi, F.A. (eds.) ILP 2010. LNCS, vol. 6489, pp. 138–145. Springer, Heidelberg (2011)
7. Nassif, H., Al-Ali, H., Khuri, S., Keirouz, W., Page, D.: An inductive logic programming approach to validate hexose binding biochemical knowledge. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 149–165. Springer, Heidelberg (2010)
8. Erickson, J.: CS 598: Computational Topology, course notes, University of Illinois at Urbana-Champaign (2009)
9. Kuželka, O., Železný, F.: Block-wise construction of acyclic relational features with monotone irreducibility and relevancy properties. In: ICML 2009: the 26th Int. Conf. on Machine Learning (2009)
10. Kuželka, O., Železný, F.: Block-wise construction of tree-like relational features with monotone reducibility and redundancy. Machine Learning 83, 163–192 (2011)
11. Krogel, M.-A., Rawles, S., Železný, F., Flach, P.A., Lavrač, N., Wrobel, S.: Comparative evaluation of approaches to propositionalization. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 197–214. Springer, Heidelberg (2003)
12. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
13. 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), 57–104 (1998)
14. Maloberti, J., Sebag, M.: Fast theta-subsumption with constraint satisfaction algorithms. Machine Learning 55(2), 137–174 (2004)
15. Atserias, A., Bulatov, A., Dalmau, V.: On the power of k-consistency. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 279–290. Springer, Heidelberg (2007)
16. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006)
17. De Raedt, L.: Logical settings for concept-learning. Artif. Intell. 95(1), 187–201 (1997)
18. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. Journal of Computer and System Sciences 64(3), 579–627 (2002)
19. Landwehr, N., Kersting, K., Raedt, L.D.: Integrating naïve bayes and FOIL. Journal of Machine Learning Research 8, 481–507 (2007)
20. Mackworth, A.: Consistency in networks of relations. Artificial Intelligence 8(1), 99–118 (1977)
21. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000-2001. Bioinformatics 17(1), 107–108 (2001)
22. Žáková, M., Železný, F., Garcia-Sedano, J.A., Masia Tissot, C., Lavrač, N., Křemen, P., Molina, J.: Relational data mining applied to virtual engineering of product designs. In: Muggleton, S.H., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 439–453. Springer, Heidelberg (2007)