Bounded Least General Generalization

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 a generalization of Plotkin's *least general generalization*. We introduce a novel concept called *bounded least general generalization w.r.t. a set of clauses* and show an instance of it for which polynomial-time reduction procedures exist. We demonstrate the practical utility of our approach in experiments on several relational learning datasets.

1 Introduction

Methods for construction of hypotheses in relational learning can be broadly classified into two large groups: methods based on specialization, so-called top-down methods, and methods based on generalization, so-called bottom-up methods. Our main motivation is to be able to learn clauses in a bottom-up manner more efficiently when we assume that there exist solutions to the learning problem from some fixed potentially infinite set. In this paper we describe a novel bottom-up method based on Plotkin's least general generalization operator [1]. We start by describing generalized versions of θ-subsumption and θ-reduction. Then we define a generalized version of least general generalization and we show that its reduced form can be computed in polynomial time w.r.t. practically relevant classes of clauses such as those with bounded treewidth. Informally, if a learning problem has a solution from a given set of clauses, such as clauses with bounded treewidth, then some equally good solution not necessarily from that set can be found using bounded least general generalizations. We demonstrate practical utility of the approach in experiments on several datasets.

2 Preliminaries: Subsumption, CSP, Treewidth

A first-order-logic clause is a universally quantified disjunction of first-order-logic literals. We treat clauses as disjunctions of literals and as sets of literals interchangeably. 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. If A and B are clauses then we say that A θ-subsumes 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 Plotkin as

F. Riguzzi and F. Železný (Eds.): ILP 2012, LNAI 7842, pp. 116–129, 2013.

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. If A is a clause and 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.

An important tool exploited in this paper, which can be used for learning clausal theories, is Plotkin's least general generalization (LGG) of clauses. A clause C is said to be a least general generalization of clauses A and B (denoted by $C = LGG(A, B)$) if and only if $C \preceq_\theta A$, $C \preceq_\theta B$ and for every clause D such that $D \preceq_\theta A$ and $D \preceq_\theta B$ it holds $D \preceq_\theta C$. A least general generalization of two clauses C, D can be computed in time $\mathcal{O}(|C| \cdot |D|)$ [2]. Least general generalization can be used as a refinement operator in searching for hypotheses [3,4]. Basically, the search can be performed by iteratively applying LGG operation on examples or on already generated LGGs. A problem of approaches based on least general generalization is that the size of a LGG of a set of examples can grow exponentially in the number of examples. In order to keep the LGGs reasonably small θ-reduction is typically applied after a new LGG of some clauses is constructed [4]. Application of θ-reduction cannot guarantee that the size of LGG would grow polynomially in the worst case, however, it is able to reduce the size of the clauses significantly in non-pathological cases.

Constraint satisfaction [5] with finite domains represents a class of problems closely related to the θ-subsumption problems. This equivalence of CSP and θ-subsumption has been exploited by Maloberti and Sebag [6] who used off-the-shelf CSP algorithms to develop a fast θ-subsumption algorithm.

Definition 1 (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 maps all variables to elements in their domains and satisfies all constraints.*

The CSP representation of the problem of deciding $A \preceq_\theta B$ has the following form. 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.

1. 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).
2. Then a relation R_l^* is constructed for every literal $l \in A$ from the arguments of literals in the respective set L_l. The relation R_l^* contains a tuple of terms $(t'_1, \ldots, t'_k)$ if and only if there is a literal $l' \in L_l$ with arguments $(t'_1, \ldots, t'_k)$.

3. 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 of terms which correspond to variables in l are retained).

Next, we exemplify this transformation process.

Example 1 (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.

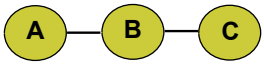
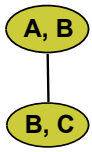
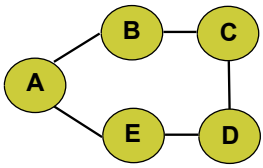
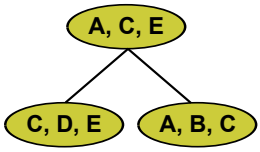
The Gaifman (or primal) graph of a clause A is the graph with one vertex for each variable $v \in vars(A)$ and an edge for every pair of variables $u, v \in vars(A)$, $u \neq v$ such that u and v appear in a literal $l \in A$. Similarly, we define Gaifman graphs for CSPs. The Gaifman graph of a CSP problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ 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 2 (Tree decomposition, Treewidth). *A tree decomposition of a graph $G = (V, E)$ is a labeled tree T such that: (i) every node of T is labeled by a non-empty subset of V, (ii) for every edge $(v, w) \in E$, there is a node of T with label containing v, w, (iii) for every vertex $v \in V$, 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.*

For example, all trees have treewidth 1, cycles have treewidth 2, rectangular $n \times n$ grid-graphs have treewidth n. Any graph with treewidth 1 is a forest. An illustration of Gaifman graphs of two exemplar clauses and their tree-decompositions is shown in Table 1. Note that tree decompositions are not unique. That is why treewidth is defined as the *maximum* cardinality of a label minus 1. Treewidth is usually used to isolate tractable sub-classes of NP-hard problems. Constraint satisfaction problems with treewidth bounded by k can be solved in polynomial time by the k-consistency algorithm[1] [8]. For constraint satisfaction problems

[1] In this paper we follow the conventions of [7]. In other works, e.g. [8], 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
$\leftarrow \mathsf{atm}(A,\mathsf{h}) \wedge$ $\mathsf{bond}(A,B,1) \wedge \mathsf{atm}(B,\mathsf{c}) \wedge$ $\mathsf{bond}(B,C,2) \wedge \mathsf{atm}(C,\mathsf{o})$		
$\leftarrow \mathsf{bond}(A,B,1) \wedge$ $\mathsf{bond}(B,C,1) \wedge \mathsf{bond}(C,D,1) \wedge$ $\mathsf{bond}(D,E,1) \wedge \mathsf{bond}(E,A,1)$		

with generally unbounded treewidth, k-consistency is only a necessary but not a sufficient condition to have a solution. If the k-consistency algorithm returns *false* for a CSP problem $\mathcal{P}$ then $\mathcal{P}$ is guaranteed to have no solutions. So, equivalently, if the problem is soluble then k-consistency always returns *true*. 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.

The following description of the k-consistency algorithm is based on the presentation by Atserias et al. [7]. 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*.

A basic property of k-consistency that we will also need is the following. If the k-consistency algorithm returns *true* for a CSP problem then it will also return *true* for any problem created from the original problem by removing some variables and some constraints, i.e. with a *subproblem*. This can be seen by noticing that if

the k-consistency algorithm starts with a set H of partial solutions and returns *true* then it must also return *true* if it starts with a superset of this set. The set of partial solutions of the subproblem must necessarily be a superset of the set of partial solutions of the original problem projected on the variables of the subproblem (from monotonicity of constraints).

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. It is known that due to this and 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 $A \preceq_\theta B$.*

Proof. Follows directly from the solubility of CSPs with bounded treewidth by the k-consistency algorithm [7] and from the equivalence of CSPs and θ-subsumption shown earlier in this section.

3 Bounded Subsumption

In this section, we introduce *bounded* versions of θ-subsumption and develop methods for working with them. We start by defining x-subsumption and x-equivalence which are weaker versions of θ-subsumption and θ-equivalence.

Definition 3 (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 w.r.t. X and $\approx_X$ is called x-equivalence w.r.t. X.*

Conventional θ-subsumption is a special case of x-subsumption. It is an x-subsumption w.r.t. the set of all clauses. It is not hard to check that x-subsumption is a transitive and reflexive relation on clauses and that x-equivalence is an equivalence-relation on clauses. Definition 3 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. However, for many practically relevant sets of clauses X, there is a relation called *x-presubsumption* which implies x-subsumption and has other useful properties as we shall see later (for example, it allows quick finding of reduced versions of clauses etc.).

Definition 4 (x-presubsumption). *Let X be a set of clauses. If $\preceq_X$ is the x-subsumption w.r.t. X and $\lhd_X$ is a relation such that: (i) if $A \lhd_X B$ and $C \subseteq A$ then $C \lhd_X B$, (ii) if $A \in X$, ϑ is a substitution and $A\vartheta \lhd_x B$ then $A \preceq_\theta B$, (iii) if $A \preceq_\theta B$ then $A \lhd_X B$. Then we say that $\lhd_X$ is an x-presubsumption w.r.t. the set X.*

The next proposition shows that if X is a set of clauses and $\lhd_X$ is an x-presubsumption w.r.t. X then $\lhd_X$ provides a sufficient condition for x-subsumption w.r.t. X.

Proposition 2. *Let X be a set of clauses. If $\preceq_X$ is x-subsumption on X and $\lhd_X$ is an x-presubsumption w.r.t. X 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 from definition of x-presubsumption. Now, we can use the second condition which gives us $C \preceq_\theta B$ (note that $C \in X$ and $C\vartheta \lhd_X B$).

We will use this proposition in the next section where we deal with bounded reduction of clauses. We will use it for showing that certain procedures which transform clauses always produce clauses which are x-equivalent w.r.t. a given set X.

In the experiments presented in this paper, we use x-presubsumption w.r.t. bounded-treewidth clauses based on k-consistency algorithm.

4 Bounded Reduction

Proposition 2 can be used to check whether two clauses are x-equivalent w.r.t. a given set of clauses X. It can be therefore used to search for clauses which are smaller than the original clause but are still x-equivalent to it. This process, which we term *x-reduction*, will be an essential tool in the bottom-up relational learning algorithm presented in this paper.

Definition 5 (x-reduction). *Let X be a set of clauses. We say that a clause $\widehat{A}$ is an x-reduction of clause A if and only if $\widehat{A} \preceq_\theta A$ and $A \preceq_X \widehat{A}$ (where $\preceq_X$ denotes x-subsumption w.r.t. X) and if this does not hold for any clause $B \subsetneq \widehat{A}$ (i.e. if there is no $B \subsetneq \widehat{A}$ such that $B \preceq_\theta A$ and $A \preceq_X B$).*

For a given clause, there may be even smaller x-equivalent clauses than its x-reductions but those might be *more specific* than the original clause which would be a problem for computing bounded least general generalizations (introduced later in this paper). There may also be multiple x-reductions differing by their lengths for a single clause.

Example 2. Let $X = \{C\}$ be the set containing just the clause $C = e(V, W) \vee e(W, X) \vee e(X, Y) \vee e(Y, Z)$. Let us have another clause $A = e(A, B) \vee e(B, C) \vee e(C, A)$ for which we want to compute its x-reduction w.r.t. the set X. We can check relatively easily (e.g. by enumerating all subsets of literals from A) that the only x-reduction of A is A itself (up to renaming of variables). However, there is also a smaller clause x-equivalent to A and that is $A' = e(X, X)$. The

x-equivalence of A and A' follows from the fact that $C \preceq_\theta A$ and $C \preceq_\theta A'$ and there is no other clause other than C in the set X. It might seem that the clauses A and A' are x-equivalent only because the set X used in this example is rather pathological but, in fact, the two clauses are also x-equivalent w.r.t. the set of all clauses with treewidth at most 1.

In order to be able to compute x-reductions, we would need to be able to decide x-subsumption. However, we very often have efficient decision procedures for x-presubsumption, but not for x-subsumption. Importantly, if there is an x-presubsumption $\lhd_X$ w.r.t. a set X decidable in polynomial time then there is a polynomial-time algorithm for computing good approximations of x-reductions. We call this algorithm *literal-elimination algorithm*.

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

The next proposition states formally the properties of the literal-elimination algorithm. It also gives a bound on the size of the reduced clause which is output of the literal-elimination algorithm.

Proposition 3. *Let us have a set X and a polynomial-time decision procedure for checking $\lhd_X$ which is an x-presubsumption w.r.t. the set X. Then, given a clause A on input, the literal-elimination algorithm finishes in polynomial time and outputs a clause $\widehat{A}$ satisfying the following conditions:*

1. *$\widehat{A} \preceq_\theta A$ and $A \preceq_X \widehat{A}$ where $\preceq_X$ is an x-subsumption w.r.t. the set X.*
2. *$|\widehat{A}| \leq |\widehat{A}_\theta|$ where $\widehat{A}_\theta$ is a θ-reduction of a subset of A's literals with maximum length.*

Proof. We start by proving $\widehat{A} \preceq_\theta A$ and $A \preceq_X \widehat{A}$. This can be shown as follows. First, $A \lhd_X A'$ holds in any step of the algorithm which also implies $A \preceq_X A'$ using Proposition 2 and consequently also $A \preceq_X \widehat{A}$ because $\widehat{A} = A'$ in the last step of the algorithm. Second, $\widehat{A} \preceq_\theta A$ because $\widehat{A} \subseteq A$. Now, we prove the second part of the proposition. What remains to be shown is that the resulting clause $\widehat{A}$ will not be bigger than $\widehat{A}_\theta$. Since $\widehat{A} \subseteq A$ and $A \preceq_\theta \widehat{A}_\theta$, it suffices to show that $\widehat{A}$ cannot be θ-reducible. Let us assume, for contradiction, that it is θ-reducible. If $\widehat{A}$ was θ-reducible, there would have to be a literal $L \in \widehat{A}$ such that $\widehat{A} \preceq_\theta \widehat{A} \setminus \{L\}$. The relation $\lhd_X$ satisfies $(A \preceq_\theta B) \Rightarrow (A \lhd_X B)$ therefore it would also have to hold $\widehat{A} \lhd_X \widehat{A} \setminus \{L\}$. However, then L should have been removed by the literal-elimination algorithm which is a contradiction with $\widehat{A}$ being output of it. The fact that the literal-elimination algorithm finishes in polynomial time follows from the fact that, for a given clause A, it calls the polynomial-time procedure for checking the relation $\lhd_X$ at most $|A|$ times (the other operations of the literal-elimination algorithm can be performed in polynomial time as well).

So, the output $\widehat{A}$ of the literal elimination algorithm has the same properties as x-reduction ($\widehat{A} \preceq_\theta A$ and $A \preceq_X \widehat{A}$) with one difference and that is that it may not be minimal in some cases, i.e. that there may be some other clause $B \subsetneq \widehat{A}$ and having these properties.

5 Bounded Least General Generalization

In this section, we show how x-reductions in general, and the literal-elimination algorithm in particular, can be used in bottom-up approaches to relational learning. We introduce a novel concept which we term *bounded least general generalization*. This new concept generalizes Plotkin's *least general generalization* of clauses.

Definition 6 (Bounded Least General Generalization). *Let X be a set of clauses. A clause B is said to be a bounded least general generalization of clauses A_1, A_2, ..., A_n w.r.t. the set X (denoted by $B = LGG_X(A_1, A_2, \ldots, A_n)$) if and only if $B \preceq_\theta A_i$ for all $i \in \{1, 2, \ldots, n\}$ and if for every other clause $C \in X$ such that $C \preceq_\theta A_i$ for all $i \in \{1, 2, \ldots, n\}$, it holds $C \preceq_\theta B$.*

The set of all bounded least general generalizations of clauses A_1, A_2, ... A_n w.r.t. a set X is a superset of the set of conventional least general generalizations of these clauses. This set of all bounded least general generalizations of clauses $A_1, A_2, \ldots A_n$ is also a subset of the set of all clauses which θ-subsume all A_1, A_2, ... A_n. There are two main advantages of bounded least general generalization over the conventional least general generalization. The first main advantage is that the reduced form of bounded least general generalization can be computed in polynomial time for many practically interesting sets X. The second main advantage is that this reduced form can actually be smaller than the reduced form of conventional least general generalization in some cases.

It is instructive to see why we defined bounded least general generalization in this particular way and not in some other, seemingly more meaningful, way. Recall that our main motivation in this paper is to be able to learn clauses more efficiently when we know (or assume) that there exist solutions to the learning problem (clauses) from some fixed potentially infinite set. Having this motivation in mind, one could argue that, for example, a more meaningful definition of bounded least general generalization should require the resulting clause to be from the set X. However, least general generalization would not exist in many cases if defined in this way, which is demonstrated in the next example.

Example 3. Let $X = \{C_1, C_2, \ldots\}$ be a set of clauses of the following form: $C_1 = e(A_1, A_2), C_2 = e(A_1, A_2) \vee e(A_2, A_3), C_3 = \ldots$. Let us also have the following two clauses: $A = e(K, L) \vee e(L, K), B = e(K, L) \vee e(L, M) \vee e(M, K)$. We would like to find a clause from X which would be their least general generalization but this is impossible for the following reason. Any clause from X θ-subsumes both A and B but none of them is least general because for any $C_i \in X$ we have $C_{i+1} \not\preceq_\theta C_i$, $C_{i+1} \preceq_\theta A$ and $C_{i+1} \preceq_\theta B$. On the other hand, bounded least

general generalization, as actually defined, always exists which follows trivially from the fact that the conventional least general generalization as computed by Plotkin's algorithm is also a bounded least general generalization.

Note that we would have to face the same problems even if X consisted of more general clauses, for example, if X consisted of clauses of treewidth bounded by k or of acyclic clauses etc, so they are not caused by some specificities of the rather artificial set X.

The reduced forms of bounded least general generalizations can often be computed in polynomial time using the literal-elimination algorithm.

Proposition 4. *Let X be a set of clauses and let $\lhd_X$ be an x-presubsumption w.r.t. the set X then the clause*

$$B_n = litelim_X(LGG(A_n, litelim_X(LGG(A_{n-1}, litelim_X(LGG(A_{n-2}, \dots)))))$$

is a bounded least general generalization of clauses A_1, A_2, ..., A_n w.r.t. X (here, $litelim_X(\dots)$ denotes calls of the literal-elimination algorithm using $\lhd_X$).

Proof. First, we show that $B \preceq_\theta A_i$ for all $i \in \{1, 2, \dots, n\}$ using induction on n. The base case $n = 1$ is obvious since then $B_1 = A_1$ and therefore $B_1 \preceq_\theta A_1$. Now, we assume that the claim holds for $n-1$ and we will show that then it must also hold for n. First, $B_n = LGG(A_n, B_{n-1})$ θ-subsumes the clauses $A_1, \dots, A_n$ which can be checked by recalling the induction hypothesis and definition of LGG. Second, $litelim_k(LGG(A_n, B_{n-1}))$ must also θ-subsume the clauses $A_1, \dots, A_n$ because $litelim_k(LGG(A_n, B_{n-1})) \subseteq LGG(A_n, B_{n-1})$.

Again using induction, we now show that $C \preceq_\theta B_n$ for any $C \in X$ which θ-subsumes all A_i where $i \in \{1, \dots, n\}$. The base case $n = 1$ is obvious since then $B_1 = A_1$ and therefore every C which θ-subsumes A_1 must also θ-subsume B_1. Now, we assume that the claim holds for $n - 1$ and we prove that it must also hold for n. That is we assume that

$$C' \preceq_\theta B_{n-1} = litelim_X(LGG(A_{n-1}, litelim_X(LGG(A_{n-2}, litelim_X(\dots)))))$$

for any $C' \in X$ which θ-subsumes the clauses A_1, A_2, ..., A_{n-1}. We show that then it must also hold $C \preceq_E B_n = litelim_X(LGG(A_n, B_{n-1}))$ for any $C \in X$ which θ-subsumes the clauses A_1, A_2, ..., A_n. We have $C \preceq_\theta LGG(A_n, B_{n-1})$ because $C \preceq_\theta B_{n-1}$ which follows from the induction hypothesis and because any clause which θ-subsumes both A_n and B_{n-1} must also θ-subsume $LGG(A_n, B_{n-1})$ (from the definition of LGG). It remains to show that C also θ-subsumes $litelim_X(LGG(A_n, B_{n-1}))$. This follows from

$$LGG(A_n, B_{n-1}) \preceq_X litelim_X(LGG(A_n, B_{n-1}))$$

(which is a consequence of Proposition 3) because if

$$LGG(A_n, B_{n-1}) \preceq_X litelim_X(LGG(A_n, B_{n-1}))$$

then

$$(C \preceq_\theta LGG(A_n, B_{n-1})) \Rightarrow (C \preceq_\theta litelim_X(LGG(A_n, B_{n-1})))$$

for any clause $C \in X$ (this is essentially the definition of x-subsumption).

One of the classes of clauses w.r.t. which the reduced forms of bounded LGGs can be computed efficiently is the class of clauses with bounded treewidth. What we need to show is that there is a polynomial-time decidable x-presubsumption relation. In the next proposition, we show that k-consistency algorithm [7] can be used to obtain such an x-presubsumption.

Proposition 5. *Let $k \in N$ and let $\lhd_k$ be a relation on clauses defined as follows: $A \lhd_k B$ if and only if the k-consistency algorithm run on the CSP-encoding (described in Section 2) of the θ-subsumption problem $A \preceq_\theta B$ returns true. The relation $\lhd_k$ is an x-presubsumption w.r.t. the set X_k of all clauses with treewidth at most k.*

Proof. We need to verify that $\lhd_k$ satisfies the conditions stated in Definition 4.

1. *If $A \lhd_k B$ and $C \subseteq A$ then $C \lhd_k B$.* This holds because if the k-consistency algorithm returns *true* for a problem then it must also return *true* for any of its subproblems (recall the discussion in Section 2). It is easy to check that if $C \subseteq A$ are clauses then the CSP problem encoding the θ-subsumption problem $C \preceq_\theta B$ is a subproblem of the CSP encoding of the θ-subsumption problem $A \preceq_\theta B$. Therefore this condition holds.
2. *If $A \in X$, ϑ is a substitution and $A\vartheta \lhd_k B$ then $A \preceq_\theta B$.* The CSP encoding of the problem $A \preceq_\theta B$ is a subproblem of the problem encoding $A\vartheta \preceq_\theta B$, in which there are additional constraints enforcing consistency with the substitution ϑ (because the set of constraints of the former is a subset of the constraints of the latter). Therefore if $A\vartheta \lhd_k B$ then also $A \lhd_k B$ and, since $A \in X$, it also holds $A \preceq_\theta B$.
3. *If $A \preceq_\theta B$ then $A \lhd_k B$.* This is a property of k-consistency (recall the discussion in Section 2).

6 Experiments

In this section, we describe experiments with a simple learning method based on three main ingredients: (i) bounded least general generalization w.r.t. clauses of treewidth 1, (ii) sampling and (iii) propositionalization. The method is very simple once we use the machinery introduced in the previous sections. It repeats the following process for each different class label K_i until a specified number of features covering different subsets of examples is reached: It samples a random set of k examples $\{e_1, \ldots, e_k\}$ with class label K_i. Then it computes clauses $C_1 = litelim(LGG_X(e_1, e_2))$, $C_2 = litelim(LGG_X(C_1, e_3))$, $\ldots$, $litelim(LGG_X(C_{k-2}, e_k))$ and stores them. The LGGs are taken w.r.t. the set X of all clauses with treewidth 1 having constants in arguments specified by a

simple language bias. In the end, it computes extensions of the stored clauses (i.e. computes the sets of examples covered by particular clauses). The constructed clauses and their extensions give rise to an attribute-value table in which attributes are clauses and the values of these attributes are Boolean values indicating whether a clause covers an example. This attribute-value table can be then used to learn an attribute-value classifier such as decision tree or random forest.

Computation of attribute-value tables is the only place where the exponential-time θ-subsumption is used. Using the rather costly θ-subsumption as a covering relation only for construction of the attribute-value table is a good compromise. From the positive side, θ-subsumption is intuitive (certainly more so than the polynomial-time x-subsumption) and also quite expressive (again, more expressive than x-subsumption) so it makes sense to use it for computing extensions. However, using θ-subsumption for reduction of clauses, for which we use the polynomial-time k-subsumption, would be much more costly.

We evaluated the novel method called Bottom in experiments with four relational datasets: Mutagenesis [9], Predictive Toxicology Challenge [10], Antimicrobial Peptides [11] and CAD [12]. The first two datasets are classical datasets used in ILP. The first dataset contains descriptions of 188 molecules labelled according to their mutagenicity. The second dataset consists of four datasets of molecules labelled according to their toxicity to female mice, male mice, female rats and male rats. The third dataset contains spatial structures of 101 peptides, which are short sequences of amino acids, labelled according to their antimicrobial activity. The last dataset contains description of class-labeled CAD documents (product structures).

Table 2. Accuracies estimated by 10-fold cross-validation using transformation-based learning with random forests and our propositionalization method (Bottom)

	Mutagenesis	PTC(FM)	PTC(MM)	PTC(FR)	PTC(MR)	Peptides	CAD
nFOIL	76.6	60.2	63.1	**67.0**	57.3	77.2	92.7
Bottom	**78.9**	**62.4**	**65.2**	59.5	**62.2**	**82.2**	**95.8**

In the experiments, we use random forest classifiers learned on attribute-value representations of the datasets constructed using our novel method. We compare the results obtained by our method with nFOIL [13] which is a state-of-the-art relational learning algorithm combining FOIL's hypothesis search and Naive Bayes classifier. Cross-validated predictive accuracies are shown in Table 2. Our method achieved higher predictive accuracies than nFOIL in all but one case. This could be attributed partly to the use of random forests instead of Naive Bayes. However, it indicates the ability of our method to construct meaningful features in reasonable time. We used sample size equal to five for all experiments. The number of features constructed for every class was set to 100. Features were always constructed only using training examples from the

given fold. Numbers of trees of random forests were selected automatically using internal cross-validation. The average runtimes of our method were: 0.3 min for Peptides, 0.6 min for CAD, 2.4 min for Mutagenesis, 4.4 min for PTC(MR), 5.1 min for PTC(FM), 9.8 min for PTC(FR) and 9.9 min for PTC(MM).

7 Related Work

The first method that used least general generalization for clause learning was Golem [3]. Golem was restricted to ij-determinate clauses in order to cope with the possibly exponential growth of LGGs. However, most practical relational learning problems are highly non-determinate. A different approach was taken in [14] where an algorithm called ProGolem was introduced. ProGolem is based on so-called asymmetric relative minimal generalizations (ARMGs) of clauses relative to a bottom clause. Size of ARMGs is bounded by the size of bottom clause so there is no exponential growth of the sizes of clauses. However, ARMGs are not unique and are not *least*-general.

Recently, an approach related to ours has been introduced [15] in which arc-consistency was used for structuring the space of graphs. There, arc-consistency was used as a covering operator called AC-projection. In contrast, we do not use the weaker versions of θ-subsumption (x-subsumptions) as covering operators in this paper but we use them only for reduction of clauses which allows us to guarantee that if a solution of standard learning problems with bounded-tree-width exists, our method is able to solve the learning problem. Thus, our approach provides theoretical guarantees which relate directly to learning problems with standard notions of covering (i.e. θ-subsumption), whereas the other approach can provide guarantees only w.r.t to the weaker (and less intuitive) AC-projection covering relation. Our framework is also more general in that it allows various different classes of clauses w.r.t. which it can compute bounded LGGs.

Another approach related to ours is the work of Horváth et al. [4] which is also based on application of least general generalization. Their approach relies on the fact that least general generalization of clauses with treewidth 1 is again a clause with treewidth 1. Since clauses with treewidth 1 can be reduced in polynomial time and since θ-subsumption problems $A \preceq_\theta B$ where A has treewidth 1 can be decided in polynomial time as well, it is possible to construct features in a manner similar to ours using only polynomial-time reduction and θ-subsumption. As in our approach, the size of the clauses constructed as least general generalizations may grow exponentially with the number of learning examples. However, unlike our approach which does not put any restriction on learning examples, the approach of Horváth et al. requires learning examples to have treewidth 1. Our approach is therefore more general even if we consider just bounded least general generalization w.r.t. the set of clauses with treewidth 1.

In a similar spirit, Schietgat et al. [16] introduced a new method based on computing maximum common subgraphs of outerplanar graphs under so-called *block-and-bridge-preserving isomorphism* which can be done in polynomial time.

This method was demonstrated to be highly competitive to *all-inclusive* strategies based on enumeration of all frequent graphs while using much lower number of maximum common subgraphs. Since it requires learning examples to be outerplanar graphs, it could not be applied to some of our datasets (e.g. the CAD dataset or the dataset of antimicrobial peptides). Aside this, another difference to our method is that it is based on a restricted form of subgraph isomorphism whereas our method is based on θ-subsumption, i.e. on homomorphism.

8 Conclusions

We have introduced a new weakened version of least general generalization of clauses which has the convenient property that its reduced form can be computed in polynomial time for practically relevant classes of clauses. Although this paper is mostly theoretical, we have also shown the practical utility of the weakened LGG in experiments where it was able to quickly find good sets of features. In our ongoing work we are developing a learning system based on the concepts presented in this paper. The system uses the bounded operations for hypothesis search and it avoids using the exponential time procedures in the learning phase altogether. Its description would not fit in the limited space available. It will be described in a longer version of this paper.

Acknowledgements. This work was supported by the Czech Grant Agency through project 103/10/1875 *Learning from Theories*.

References

1. Plotkin, G.: A note on inductive generalization. Edinburgh University Press (1970)
2. Nienhuys-Cheng, S.-H., de Wolf, R. (eds.): Foundations of Inductive Logic Programming. LNCS, vol. 1228. Springer, Heidelberg (1997)
3. Muggleton, S., Feng, C.: Efficient induction of logic programs. In: ALT, pp. 368–381 (1990)
4. Horváth, T., Paass, G., Reichartz, F., Wrobel, S.: A logic-based approach to relation extraction from texts. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 34–48. Springer, Heidelberg (2010)
5. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
6. Maloberti, J., Sebag, M.: Fast theta-subsumption with constraint satisfaction algorithms. Machine Learning 55(2), 137–174 (2004)
7. 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)
8. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006)
9. Srinivasan, A., Muggleton, S.H.: Mutagenesis: ILP experiments in a nondeterminate biological domain. In: ILP, pp. 217–232 (1994)
10. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000-2001. Bioinformatics 17(1), 107–108 (2001)

11. Cherkasov, A., Jankovic, B.: Application of 'inductive' qsar descriptors for quantification of antibacterial activity of cationic polypeptides. Molecules 9(12), 1034–1052 (2004)
12. 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)
13. Landwehr, N., Kersting, K., De Raedt, L.: Integrating naïve bayes and FOIL. Journal of Machine Learning Research 8, 481–507 (2007)
14. Muggleton, S., Santos, J., Tamaddoni-Nezhad, A.: Progolem: A system based on relative minimal generalisation. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 131–148. Springer, Heidelberg (2010)
15. Liquiere, M.: Arc consistency projection: A new generalization relation for graphs. In: Priss, U., Polovina, S., Hill, R. (eds.) ICCS 2007. LNCS (LNAI), vol. 4604, pp. 333–346. Springer, Heidelberg (2007)
16. Schietgat, L., Costa, F., Ramon, J., De Raedt, L.: Effective feature construction by maximum common subgraph sampling. Machine Learning 83(2), 137–161 (2011)