HiFi: Tractable Propositionalization through Hierarchical Feature Construction

Ondřej Kuželka and Filip Železný

Intelligent Data Analysis Research Group
Dept. of Cybernetics, Czech Technical University in Prague
http://ida.felk.cvut.cz
{kuzelo1,zelezny}@fel.cvut.cz

Abstract. We present a novel propositionalization algorithm HiFi based on constructing first-order features with hierarchical structure. Unlike state-of-the-art algorithms, HiFi simultaneously constructs features and computes their extensions, while this merged operation takes time polynomial in the maximum feature length and in the total number of features. Moreover, all features produced by HiFi are the smallest in their semantic equivalence class. In our preliminary experiments we show runtime improvements with respect to a state-of-the-art propositionalization algorithm.

1 Introduction

This paper addresses the problem of propositionalization, i.e. converting structured descriptions of learning examples into attribute-value descriptions. A major stream of state-of-the-art approaches to propositionalization [3, 2, 5] is based on constructing *features* in the form of queries. A set of generated features then plays the role of the attribute set for the new representation. The range of all possible features is, for a given learning problem, usually constrained by declaring a language of admissible features. However, this popular form of propositionalization suffers from two sources of complexity: i) finding features complying to the specified language constraints and ii) computing extension of the generated features, i.e. determining for which examples they are true. These sub-problems usually introduce two exponential (in n) time complexity factors into propositionalization (RSD [5] or SINUS [2]). This paper mainly shows how both of these intractability sources can be simultaneously removed (reduced to polynomial-time) if one only works with *hierarchical features*.

In [7] it was shown that certain constraints imposed on the feature language enable to reduce the feature construction problem to the problem of satisfying a (polynomially large) set of propositional Horn clauses, which is a tractable problem. One of such sufficient constraints is the hierarchical structure of features. In this work we additionally remove the second source of complexity (finding extensions) for hierarchical features. For this we exploit the fact that verifying subsumption between a feature and an example can be reduced to a constraint

satisfaction problem, which in case of a hierarchical feature can be efficiently solved using a method known as directed arc consistency checking.

It would be thus theoretically straightforward to implement a propositionalization algorithm by two subsequent problem reductions (HornSAT and CSP). This would be impractical namely due to the overhead incurred by representation conversions and by the implied separation of feature construction from extension computation. HiFi works in the polynomial bound but brings further benefits by avoiding explicit reductions and by merging the mentioned two stages of propositionalization. A further advantage of HiFi is that it only produces *reduced* features; i.e. of all semantically equivalent features it only produces the smallest. We omit proofs, which will appear in an extended account of this work

2 Preliminaries

Most algorithms in inductive logic programming rely on a generality relation between clauses. While it would be natural to say that C is more general than D whenever C entails D (written $C \models D$), the entailment relation is undecidable and thus is typically approximated by θ-subsumption.

Definition 1 (Subsumption, equivalence, reduction). *Let C and D be clauses and let there be a substitution θ such that $lits(C\theta) \subseteq lits(D)$. We say that C θ-subsumes D (written $C \preceq_\theta D$). If further $D \preceq_\theta C$, we call C and D θ-equivalent (written $C \approx_\theta D$). We say that C is θ-reducible if there exists a clause C' such that $C \approx_\theta C'$ and $|C| > |C'|$. A clause C' is said to be a θ-reduction of C if $C \approx_\theta C'$ and C' is not θ-reducible.*

It is a NP-complete problem to decide θ-subsumption between two clauses and co-NP-complete to decide θ-reduction [4].

Following up on established approaches to propositionalization, we further constrain the language bias for features. We do this through the notion of a feature template.

Definition 2 (Template). *A* template *τ is a pair (γ, μ) where γ is a ground function-free query and μ is a subset of all arguments in τ. Arguments of τ contained in μ are called* input arguments*, the other arguments are called* output arguments*. Let clause $C \preceq_\theta \gamma$. An occurence of variable v at the i-th argument of literal l in C is called an* input occurrence *(w.r.t. τ and θ) if the i-th argument of literal $\lambda_{C,\gamma,\theta}(l)$ is an input argument, otherwise it is an* output occurrence *(w.r.t. τ and θ). A literal in C containing only output variables is called a* root *of C (w.r.t. τ and θ).*

Definition 3 (Feature). *Let $\tau = (\gamma, \mu)$ be a template and $n \in N$. A query f containing no constants or functions is a* feature *correct w.r.t τ and n if $|f| \leq n$, $f \preceq_\theta \gamma$ and for every input (output, respectively) occurrence of a variable in f there is also an output (input) occurrence of the same variable in f, where both occurrences are taken w.r.t τ and θ.*

Note that a θ-reduction of a correct feature may not be a correct feature itself. This may represent a problem because, to avoid redundancy, we would like to work with reduced features. In the next section we will show how to reduce *hierarchical* features while maintaining correctness.

3 Hierarchical Features

In this section, we define hierarchical features and list some of their properties regarding θ-reduction and CSP representations.

Definition 4 (Hierarchical feature). *A template $\tau = (\gamma, \mu)$ is* hierarchical *if every literal in γ has at most one input argument and there is a partical irreflexive order $\prec$ on constants in γ such that $c \prec c'$ whenever c (c') occurs at an input (output) argument of γ. A feature correct w.r.t. a hierarchical template is a* hierarchical feature *if it has exactly one root.*

Graphically, a hierarchical feature f corresponds to a unique tree graph T_f with vertices v_i corresponding to literals l_i. There is an edge between v_i and v_j iff a variable has an output occurrence in l_i and an input occurrence in l_j. Consider a subtree S_f of T_f. A query q consisting of literals corresponding to vertices in S_f is called a *subfeature* of f, and q's literal corresponding to the root of S_f is its *subroot* (w.r.t. f).

The next definition introduces $H\theta$-subsumption and $H\theta$-reduction. Informally, $H\theta$-subsumption requires that θ maps literals at a given depth in one clause to literals at the same depth in the other clause.

Definition 5 ($H\theta$-reduction). *We say that hierarchical feature f $H\theta$-subsumes hierarchical feature g (written $f \preceq_{H\theta} g$) iff there is a substitution θ such that $f\theta \subseteq g$ and for every literal $l \in lits(f)$ there is a literal $l\theta \in lits(g)$ such that $depth_f(l) = depth_g(l)$. If further $g \preceq_{H\theta} f$, we call f and g $H\theta$-equivalent (written $f \approx_{H\theta} g$). We say that f is $H\theta$-reducible if there is a hierarchical feature f' such that $f \approx_{H\theta} f'$ and $|f| > |f'|$. Hierarchical feature f' is said to be a $H\theta$-reduction of f if $C \approx_{H\theta} f'$ and f is not $H\theta$-reducible.*

Clearly, $H\theta$-reducibility implies θ-reducibility, but not vice versa. The next lemma provides the basic means to detect $H\theta$-reducibility of a hierarchical feature.

Lemma 1. *A hierarchical feature f is $H\theta$-reducible if and only if it has subfeatures f_1, f_2 whose respective subroots share the same input variable, and $f_1 \preceq_{H\theta} f_2$.*

As a consequence of the lemma above, it is possible to decide reducibility of a hierarchical feature f in time polynomial in the size of f.

Lemma 2. *A constraint graph induced by a subsumption problem $f \preceq_\theta e$ where f is a hierarchical feature, has constraints only over those pairs of variables corresponding to literal pairs where the shared FOL variable is used as an output in one of them and as an input in the other one.*

Algorithm 1 *computeDomain*$(root, example)$

```
1: Input: Root of the constraint graph root, Example e;
2: rootDomain ← { all literals built on the same predicate symbol }
3: for ∀child ∈ children(root) do
4:     childDomain ← computeDomain(child)
5:     remove all values from rootDomain for which there are no values in childDomain such that
       the corresponding constraint would be satisfied
6: end for
   return rootDomain
```

Lemma 2 allows us to reduce the problem of deciding θ-subsumption for hierarchical features to problem of solving a tree-structured constraint satisfaction problem. It is known that such CSP problems are efficiently soluble [1]. Here we follow an algorithm based on directed-arc-consistency. The basic idea of the directed-arc-consistency method is that when we filter all domains of CSP-variables in such a way that domains of these variables contain only values consistent with filtered domains of their children, then it is possible to find a solution by assigning values from the filtered domains to CSP-variables proceeding from root to leaves (Algorithm 1).

4 The Propositionalization Algorithm

In this section, we design a propositionalization algorithm HiFi, which runs in time polynomial in the maximum feature size and in the number of generated features. HiFi merges the two usual phases of propositionalization, i.e. feature construction and extension computation. Specifically, the core algorithm accepts a learning example and a feature template. It produces all features complying to the template and subsuming the example. These features are obtained by combinatorial composion of subfeatures, which act as the primitive building blocks. One of the advantages of this assembly approach is that subsumption of the given example can already be checked for individual subfeatures; if it is refuted for a given subfeature, this subfeature is not used in the subsequent feature assembly.

The algorithm exploits the partial irreflexive order, which is imposed on types of arguments by Def. 4. Due to existence of this order it is possible to sort all declared predicates $l \in \gamma$ topologically with respect to a graph induced by the partial order. When the topological ordering is found, it is possible to organize generation of features in such a way that subfeatures are built iteratively by combining smaller subfeatures into larger ones. We illustrate this through an example.

Example 1. Consider the following template

$$\tau = car(-c) \wedge load(+c, -l) \wedge triangle(+l) \wedge box(+l).$$

The toplogical order of literals $l \in \tau$ then corresponds to $(box(+l), triangle(+l), load(+c, -l), car(-c))$. Now, we take the first declared literal $box(+l)$ and build

the set of all possible subfeatures with $box(+l)$ as its subroot $S_{box(+l)} = \{box(L)\}$. Similarly, for $triangle(+l)$ we have $S_{triangle(+l)} = \{triangle(L)\}$. The third declared predicate $load(+c, -l)$ has outputs. Thus, subfeatures with this predicate in the subroots are obtained by all possible graftings of the already generated subfeatures onto the $load/2$ subroot. This results in

$$S_{load(+c,-l)} = \{load(C, L) \wedge triangle(L), load(C, L) \wedge box(L), \ldots$$

$$\ldots load(C, L) \wedge box(L) \wedge triangle(L)\}.$$

The set $S_{car(-c)}$ is then created similarly as $S_{load(+c,-l)}$. Note that so far we have not considered that a maximum allowed size of a feature is specified.

Generating features in this manner can be conveniently combined with computation of θ-subsumption. Brief inspection of Algorithm 1 reveals that in order to compute domain of a literal, which is a subroot of some subfeature, we only need to know the domains of its children. However, the domain of any literal l can be computed when l is added as a subroot of some subfeature to the set of already generated subfeatures and then it can be reused many times. Due to Lemma 1 we can also decide $H\theta$-reducibility for any subfeature s in polynomial time and remove s from the set of already generated features if it is found reducible. What remains to explain is how HiFi deals with the maximum declared feature size, which is answered by the following lemma.

Lemma 3. *Let m denote number of declared predicates and let a denote maximum arity of the predicates. Then, for all declared predicates p, we can find sizes of the smallest features F_{min} containing p in time $O(m^2 + m \cdot a)$.*

Due to Lemma 3, which allows us to decide whether a subfeature may be extended to a correct feature with size less than n, and due to the fact that no feature f has more than n subfeatures, the number of generated subfeatures can be bounded at any time of HiFi's run by $n \cdot C(n)$, where $C(n)$ is the total number of correct features w.r.t. τ. Brief combinatorial reasoning then implies that HiFi indeed runs in time polynomial in the maximum feature length and in the total number of features.

Theorem 1. *Let τ be a hierarchical template and n be maximum feature size, then HiFi finishes in time polynomial in the total number of features and in n.*

5 Experiments

Here we evaluate HiFI in comparison to a state-of-the-art propositionalization algorithm RSD [5]. Due to limited space, we perform experiments only in one relational domain. In this experiment the same language bias is applied for HiFi and RSD. The experiments pertain to class-labeled CAD data (product structures) described in [6], consisting of 96 CAD examples each containing several hundreds of first-order literals. Table 1 displays the results. J48 refers to leave-one-out predictive accuracies of J48 decision trees built using the generated

features. The results indicate that for large features HiFi outperforms RSD by several orders of magnitude. However, for very small features HiFi's more complicated algorithms cause some overhead, which manifests itself in RSD being faster for small sizes.

Length	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
HiFi [s]	12	14	15	15	30	53	115	261	618
RSD [s]	0.5	1.5	8	45	310	1749	12324	n.a.	n.a.
J48 [%]	88.54	87.5	94.79	93.75	95.83	94.79	91.66	91.66	92.7

Table 1. Propositionalization results on CAD data.

6 Conclusions

We have presented a novel propositionalization algorithm HiFi based on constructing first-order features with hierarchical structure. Our experiments indicate that HiFi performs substantially faster than state-of-the-art propositionalization system RSD [5]. Experiments with more datasets and comparing HiFi to other ILP algorithms should be done in future work.

Acknowledgements
This work is supported by the Grant Agency of the Czech Republic through the project 201/08/0486 Merging Machine Learning with Constraint Satisfaction.

References

1. R. Barták. Theory and practice of constraint propagation. In *Proceedings of the 3rd Workshop on Constraint Programming for Decision and Control (CPDC2001)*, pages 7–14, 2001.
2. M.-A. Krogel, S. Rawles, F. Železný, P. A. Flach, N. Lavrač, and S. Wrobel. Comparative evaluation of approaches to propositionalization. In *Procs. of the 13th International Conf. on Inductive Logic Programming*, pages 197–214, 2003.
3. N. Lavrač and P. A. Flach. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, 2(4):458–494, 2001.
4. J. Maloberti and E. Suzuki. An efficient algorithm for reducing clauses based on constraint satisfaction techniques. In *ILP*, volume 3194 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2004.
5. F. Železný and N. Lavrač. Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62(1-2):33–63, 2006.
6. M. Žáková, F. Železný, J. Garcia-Sedano, C. Masia Tissot, N. Lavrač, P. Křemen, and J. Molina. Relational data mining applied to virtual engineering of product designs. In *Procs of the 16th Int. Conference on Inductive Logic Programming*, volume 4455 of *LNAI*, pages 439–453. Springer, 2007.
7. F. Železný. Efficient sampling in relational feature spaces. In *Proceedings of the 15th Int. Conf. on Inductive Logic Programming*, pages 397–413. Springer, 2005.