Efficient Sampling in Relational Feature Spaces

Filip Železný

Czech Technical University in Prague,
Technická 6, 166 27, Prague 6, Czech Republic
zelezny@fel.cvut.cz

Abstract. State-of-the-art algorithms implementing the 'extended transformation approach' to propositionalization use backtrack depth first search for the construction of relational features (first order atom conjunctions) complying to user's mode/type declarations and a few basic syntactic conditions. As such they incur a complexity factor exponential in the maximum allowed feature size. Here I present an alternative based on an efficient reduction of the feature construction problem on the propositional satisfiability (SAT) problem, such that the latter involves only Horn clauses and is therefore tractable: a model to a propositional Horn theory can be found without backtracking in time linear in the number of literals contained. This reduction allows to either efficiently enumerate the complete set of correct features (if their total number is polynomial in the maximum feature size), or otherwise efficiently obtain a random sample from the uniform distribution on the feature space. The proposed sampling method can also efficiently provide an unbiased estimate of the total number of correct features entailed by the user language declaration.

1 Introduction

A major stream of approaches to propositionalization [6] is based on constructing relational features in the form of Datalog queries, such as the one below

$$\texttt{car(C)} \wedge \texttt{load(C,L)} \wedge \texttt{small(L)} \wedge \texttt{triangle(L)}$$

from the well-known Michalski's east-west trains domain, querying whether there is car carrying a small, triangle shaped load (in a train). In this paper I constrain myself to expressions that are conjunctions of non-negated atoms without constants (thus avoiding atoms such as `numOfWheels(Car, 2)` and rather considering an atom `has2Wheels(Car)`). Much like traditional ILP systems suffer from two sources of computational complexity–the size of the hypothesis space and the complexity of proving examples from a hypothesis–the burden of this propositionalization approach is also twofold, represented by these factors:

1. the complexity of constructing a syntactically well-formed feature definition
2. the complexity of finding the extension of a feature, ie. the subset of data instances for which the feature holds true.

A lot of research has been conducted to make subsumption check based proving (ie. the problem underlying Item 2) more efficient. This includes both enhancements preserving completeness and correctness [2] as well as those representing tractable approximations to the subsumption check [11]. In contrast, Item 1 is in state-of-the-art propositionalization systems approached through an exhaustive, usually depth-first search, which of course becomes quickly intractable once language bounds (eg. the maximum number of atoms in a feature) are softened.

Note the complexity trade-off between Items 1 and 2. The finer conditions are stipulated on the acceptable syntactical form of a feature, the fewer correct features exist in the search space, decreasing the effort needed to exert in Item 2, but the more difficult it may be to find a correct feature if one resorts to a naive backtrack search.

Correct Feature. In this paper, I constrain the notion of a well-formed feature[1] in a natural way, by combining two popular language-bias specification techniques. First, as in many successful ILP systems (such as Progol [8]), I assume the user to pre-specify the set of predicates which can be employed in a feature, as well as *types* and *modes* of each argument place therein. In a correct feature, no variable appears at two, differently typed arguments. For each argument, the mode is either '+', or '−' and a variable occurrence at that argument is called an input, or output, respectively. Furthermore, the maximum *branching factor* (maximum number of occurrences of a given predicate in a feature with the same input variables[2]), and the maximum *size* of a feature (maximum number of atoms contained), are pre-set. Second, I impose the provisos suggested in the Extended Transformation Approach propositionalization framework [7], namely that (i) each variable in a feature is used exactly once as an output and at least once as an input, (ii) no correct feature is an atom-wise union of two or more correct features.

Conditions (i) and (ii) actually distinguish the feature construction process from the clause-enumeration procedures at the heart of most ILP systems. While (i) is motivated primarily by the ease of human interpretation of a relational feature, (ii) prevents the assembly of features by simply conjoining simpler ones–an excess expressivity given that propositional algorithms, to which the resulting features are subjected, are themselves able to construct conjunctions.

Assume the user declaration is specified, including n, β representing the bounds on the feature size and branching-factor, respectively. Here I mainly show that if the declaration obeys certain easily acceptable restrictions, one can either efficiently (in time polynomial in n and β) enumerate the complete set of correct features (if the number N of actually existing correct features is polynomial in n), or efficiently obtain a polynomial-size random sample of correct features (if N is exponential in n) from a uniform probability distribution on the set of all correct features. Although it is not known beforehand, whether or not N is polynomial in n given a declaration and varying n, running the two respective

[1] I will use interchangeably the terms 'feature', 'correct feature' and 'well-formed feature'.

[2] This parameter is called *recall* in Progol.

algorithms in parallel would of course result in obtaining one of the two results efficiently. The fundamental technique I exploit is a polynomial-time reduction of the feature-construction problem onto an instance of HORN-SAT, ie. finding a model of a propositional Horn theory. It is interesting to note that HORN-SAT is the only non-trivial tractable subclass of the generally NP-complete SAT problem [10]. A model to a propositional Horn theory can be found without backtracking, in time linear in the number of literals in the theory [5].

Let me now present three specific reasons why the method here presented is an important contribution to both propositionalization and state-of-the-art relational learning in general.

1. The first reason is practical. Even if a user requires to obtain the complete set of correct features rather than a sample, and thus resorts to an exhaustive enumerative method, there is presently no way of efficiently determining how large a set of correct features is entailed by the current language declaration. Thus the typical propositionalization modus operandi consists of repeated executions of the feature construction process stopped after a long run time and an unacceptable number of features generated, followed by iterative re-tuning of the declaration.[3] A consequence of my sampling method is that the total number of correct features can be efficiently and accurately estimated prior to enumerating all features.
2. Recent research [14,4,1] indicates the possibility that rather than using an exhaustive set of features enumerated from a small space of simple expressions, it may be beneficial to uniformly sample (eg. the same number of) features from a larger space (for which an exhaustive method is intractable), allowing for more descriptive complexity as well as variability between the features. My method provides the necessary bits for this sake.
3. A recent, very interesting paper [13] shows the advantages of constructing random-forest classifiers based on relational features. A random sample of features provided by the method presented here can be used as an input to construct a randomized decision tree, as a component of a random forest.[4] Experimental evaluation of this idea is however out of the scope of this paper.

2 Correct Features as HORN-SAT Solutions

Before exposing details, here is a brief outline of my strategy. Recall the definition of a *correct feature* from the introduction. Let me call a finite set of constant-free Datalog atoms[5] an expression and, given a mode/type declaration, let every expression be called *proper* if all its variables have exactly one output and at least one input occurrence, and *connected* if it all its atoms are pairwise connected.

[3] This argument of course follows solely from my subjective experience.

[4] My method only generates one of two types of features considered in the mentioned paper, there called *selective* features.

[5] For simplicity I work with sets, although examples of such expressions will be shown as conjunctions of the elements.

Two atoms a and b in an expression are connected if they share a variable or both a and b are connected with another atom c in the expression. Clearly, any correct feature is connected (remind the requirement of the undecomposability of a feature into two or more features). In this section, the adjective *polynomial (exponential)* will stand for *polynomial (exponential) in* n (the maximum feature size). By definition of the branching factor β, it must hold $\beta \leq n$, so for simplicity of analysis, I will use the upper-bound $\beta := n$ and then make sure that the time complexity of the algorithm is polynomial in n implying it is also polynomial in β.

I first construct a 'bottom feature' $\perp$ – a proper expression complying to the type/mode declaration, which is an atom-wise superset (up to variable renaming) of all correct features. Slight restrictions on the user declaration will guarantee that $\perp$ exists, has a polynomial number of atoms, and can be constructed in polynomial time. As $\perp$ complies to typing and moding constraints, so do all its subsets. What remains to do then is to find all proper, connected subexpressions of $\perp$ of size $\leq n$, A straightforward verification of all subexpression of size $\leq n$ would obviously require exponential time, however, I show that this problem may be efficiently reduced onto a polynomial-size HORN-SAT instance, for which an efficient solving algorithm exists.

2.1 Bottom Feature Construction

I now regard the first step, ie. constructing the bottom feature $\perp$ given a type/mode declaration and n, the maximum feature size. To encode a declaration, I employ a simple form used with slight variations in numerous ILP systems. Here, available predicates are listed with mode and type indicators plugged into the argument places. An example declaration follows

```
car(-c), hasRoof(+c), load(+c,-l), triangle(+l), box(+l)
```

The modes `-/+` denote outputs/inputs, respectively, and `c`, `l` represent the respective car and load argument types. I now impose two natural, yet important restrictions on declarations. First, a declaration has a finite size and each declared predicate has a finite arity. Second, there exists a partial irreflexive order $\prec$ on types, such that for any two types t_1, t_2 it holds $t_1 \prec t_2$ whenever t_1 occurs at an input position of a declared predicate and t_2 appears at an output position in the same predicate. This assumption is trivially met by the example declaration above (here `c` $\prec$ `l`). The declaration would remain valid if eg. `tows(+c,+c)` was added to it, but not if `tows(+c,-c)` was added. Finally, for clarity of explanation I will only consider predicates with at most one output argument, although the further presented principles do not require that condition.

To demonstrate the construction of $\perp$, I will distinguish two cases: (SI) any declared predicate has at most one input, (MI) some have two or more inputs. I will first exemplify the former case, using the sample declaration above. Due to the $\prec$ existence and the assumption (SI), every correct feature can be represented as a tree, where vertices correspond to atoms and edges connect pairs of atoms where one contains a variable as an output and the other contains the same

$$\beta \le n \left\{ \underbrace{\begin{array}{llll} \texttt{car(C)} & -\texttt{hasRoof(C)} & & \\ & -\texttt{load(C,L1)} & -\texttt{box(L1)} & \\ & & -\texttt{triangle(L1)} & \\ & -\texttt{load(C,L2)} & -\texttt{box(L2)} & \\ & & -\texttt{triangle(L2)} & \\ & \vdots & \vdots & \\ & -\texttt{load(C,Ln)} & \vdots & \end{array}}_{\Delta\le\Delta_{max}} \right.$$

Fig. 1. A tree graph representing the bottom feature $\perp$ whose size is a polynomial function of n. Vertices correspond to atoms in $\perp$. β denotes the branching factor, Δ stands for the tree depth, bounded by some constant Δ_{max}.

variable as an input. Similarly, $\perp$ also corresponds to a tree, which must contain all correct features as root-sharing subtrees. The tree form of $\perp$, whose size depends on n, is sketched in Fig. 1.

Due to the feature connectivity requirement and assumption (SI), no correct feature may regard two or more cars: such an expression would necessarily be disconnected.[6] Therefore, only one `car/1` atom is present in $\perp$, as the root. Due to the assumed partial ordering of types $\prec$ and the finiteness of the declaration, the depth of the tree is bounded by some constant Δ_{max}. Also its branching factor can be upper-bounded by n (eg. no feature of size at most n can address more loads than n; this upper bound may of course be quite easily improved). The number of nodes, ie. the size of the bottom set is thus of order $n^{\Delta_{max}}$, ie. polynomial.

Consider now the more general (MI) case where a declaration contains a predicate with multiple inputs. This is a natural case in domains where a feature may relate two substructures of the individual. An example declaration follows capturing a simplified version of the Mutagenesis problem [12].

```
atm(-a), crb(+a), nit(+a), oxy(+a), hyd(+a),
   bond(+a,+a,-b), single(+b), double(+b)
```

With respect to the graph representation I introduced in the previous paragraph, due to the presence of the `bond/3` predicate with 2 inputs, correct features no longer form a tree and neither does $\perp$.[7] The proof of $\perp$ still having a polynomial size now relies on the fact that $\perp$'s atoms can still be organized in 'layers' (corresponding to the columns in Fig. 2 on Page 404), using the assumed partial type order $\prec$. Up to n atoms are in the first layer, so the first layer generates $O(n)$ output variables. The cardinality of the second layer is thus $O(n^I)$, where

[6] I ignore the case when the declaration has more than one predicate with only output variables (such as `car/1`), while assuming (SI): again due to the connectivity requirement, this case can be treated as two separate feature construction problems.

[7] Also, the `atm/1` predicate will need to be placed n times in $\perp$ with distinct output variables, unlike the `car/1` predicate in the (SI) case.

I is the maximum number of input arguments in an atom, among atoms in the declaration. The third layer may use $O(n^I)$ variables, so its cardinality is at most $O(n^{I \times I})$. Thus in general, $O(|\bot|)$ may be upper bounded by $n^{I^{\Delta_{max}}}$. Since the exponential factor is a constant, I accept this as a polynomial bound as I intended to achieve in this section.[8]

2.2 Avoiding Improperness

Having constructed $\bot$, I now proceed to the problem of how to efficiently extract $\bot$'s proper subexpressions of size $\leq n$. The basic idea is to assign a propositional variable to each atom of $\bot$ and use the variables to construct a clause set, encoding the properness requirements, so that every solution of the clause set corresponds to a proper expression. Interestingly, encoding the constraints turns out to require only Horn clauses. Again, I will illustrate the procedure by way of example in the East-West train domain, continuing with its previous predicate declaration. Let $n = 3$. Then $\bot =$

$$\underset{P_1}{\texttt{car(C)}} \wedge \underset{P_2}{\texttt{hasRoof(C)}} \wedge \underset{P_3}{\texttt{load(C,L)}} \wedge \underset{P_4}{\texttt{triangle(L)}} \wedge \underset{P_5}{\texttt{box(L)}}$$

is a correct bottom feature. Note that using the branching-factor upper-bound $\beta := n$ used above for bounding $|\bot|$, I would include three `load/2` atoms into $\bot$ (refer to the corresponding branches in Fig. 1), however, in this case all correct features of length up to 3 atoms are clearly subsets (up to variable renaming) of this shorter $\bot$. As the lower line indicates, I assign one propositional variable (P_1 to P_5) to each atom. A truth assignment to these variables will represent a $\bot$'s subexpression as follows: if and only if a variable has the *false* value, the corresponding atom *belongs* to the subexpression.[9] As the reader will easily verify, the following set of clauses is satisfied if and only if each variable present in the subexpression has at least one input occurrence (the first clause relating to `C`, the second to `L`).

$$\neg P_2 \vee \neg P_3 \vee P_1 \tag{1}$$
$$\neg P_4 \vee \neg P_5 \vee P_3 \tag{2}$$

In each clause, I introduced a negative literal corresponding to each atom containing the respective variable as an input, and the positive literal in each clause corresponds to the atom with an output appearance of the respective variable. Since I assume each variable to have exactly one output occurrence, I necessarily obtain Horn clauses. It of course remains to make sure that the mentioned

[8] This, in general rapid polynomial growth of $|\bot|$ may be reduced by imposing a small branching factor bound β.

[9] With this choise the dual propositional problem will acquire a HORNSAT form. I may equally have assigned the *true* value to denote the membership thus arriving instead at a NON-HORNSAT problem, ie. one with at most one negative literal in each clause.

assumption is indeed satisfied. Note first that by construction of $\perp$ (refer to Fig. 1), each output argument is assigned a distinct variable and therefore each variable in any subexpression of $\perp$ appears as an output *at most* once. I now need to make sure that it appears as an output *at least* once. Evidently, this is the case if and only if the following four clauses, which I add to the constructed clause set, are satisfied (the upper two for `C`, the lower two for `L`).

$$\neg P_1 \vee P_2 \qquad \neg P_1 \vee P_3 \tag{3}$$

$$\neg P_3 \vee P_4 \qquad \neg P_3 \vee P_5 \tag{4}$$

Here the negative (positive) literals correspond to input (output) occurrences of the respective variables in $\perp$. Since, as the reader has already seen, there is at most one output occurrence of each variable, also these clauses are necessarily Horn. Let me now determine the total number of Horn clauses obtained in general by the procedure so far. A simple insight yields that I get one clause per every output argument in $\perp$ (such as the two clauses 1 - 2) and one clause per every input in $\perp$ (such as the four clauses 3 - 4). Due to assigning a single propositional variable to every atom in $\perp$, the number of literals in each clause is at most $|\perp|$. As I have constructed a polynomial size $\perp$, the resulting HORN-SAT instance (consisting of all clauses 1 - 4) has a polynomial number of clauses with a polynomial number of literals in each. A trivial solution simply makes true all involved propositional variables (note the omnipresence of a positive literal). To avoid this useless solution–corresponding to the empty feature–I append one more Horn clause

$$\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_5 \tag{5}$$

2.3 Avoiding Disconnected Features

At this stage, whenever a solution satisfying all clauses constructed so far makes false n or fewer of the propositional variables, it corresponds to a correct feature. Although I constructed no dedicated clauses guaranteeing connectedness of extracted expressions, due to the (SI) character of the particular example at hand, this property is satisfied automatically: from Fig. 1 it is easy to see that any disconnected subgraph of the tree would represent an expression with an input variable with no output occurrence. Such a non-proper expression would be eliminated by the so-far constructed clauses. However, the (MI) setting allows for proper yet disconnected expressions such as

$$\texttt{atm(A)} \wedge \texttt{crb(A)} \wedge \texttt{atm(B)} \wedge \texttt{oxy(B)}$$

–an example taken from the Mutagenesis domain. The method to avoid obtaining disconnected expressions such as the above, is based on a polynomial extension of the generated Horn set. Let a *primary predicate (atom)* be a declared predicate (atom based thereon) with no input argument (ie. one relating directly to the individual, such as `atm/1` or `car/1`). I will distinguish two (MI) subcases: (MI-S) only one primary predicate is declared, and (MI-M) more than one primary predicates are declared.

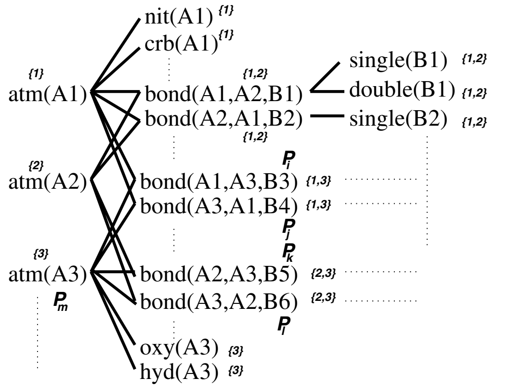

Fig. 2. A fraction of the $\perp$ graph representation in a (MI) setting. Chemical bonds are not oriented so half of the shown `bond/3` atoms are semantically superfluous, however, I am not concerned here with feature semantics.

I first regard (MI-S). In the layer-wise construction of $\perp$ (enabled again by assuming the $\prec$ order, refer to Fig. 2), I assign a distinct integer singleton label to each (necessarily primary) atom in the first layer. Then the label of every vertex in layer l ($l \geq 2$) is the union of the labels of its parents in layer $l-1$. Now I am able to identify atoms ('joints'), which are simultaneously descendants of more than one primary atoms. The idea now is to allow the inclusion of a second or further primary atom in a feature, only if there is a joint for it with another primary atom in the feature. For example, for the primary atom `atm(A3)`, I can facilitate that by adding the following Horn clause into the HORN-SAT instance (refer to the $P_m \ldots P_l$ variables assigned to vertices in the figure)

$$P_m \vee \neg P_i \vee \neg P_j \vee \neg P_k \vee \neg P_l \tag{6}$$

In general, assume that $P_1 \ldots P_{|\perp|}$ are variables assigned to atoms in $\perp$, and let $L(P_s)$ ($1 \leq s \leq |\perp|$) be a function yielding the label of the vertex corresponding to the to variable P_s. For each primary atom, corresponding to variable P_r and having the label $\{\lambda\} = L(P_r)$, the following Horn clause will be added to the HORN-SAT instance:

$$P_r \qquad \bigvee_{1 \leq s \leq |\perp|,\ \{\lambda\} \subset L(P_s),\ \forall \rho \in L(P_s):\ \rho \leq \lambda} \qquad \neg P_s \tag{7}$$

This extension has a polynomial size since it generates at most $|\perp|$ additional clauses each with at most $|\perp|$ literals. The $\rho \leq \lambda$ inequality in the selector line exploits the order imposed on primary atoms by the vertex labelling (I reflect here the order also in output-variable naming). It prevents the construction of disconnected expressions such as

```
atm(A1)∧atm(A2)∧bond(A1,A2,B1)∧single(B1)
∧atm(A3)∧atm(A4)∧bond(A3,A4,B2)∧single(B2)
```

This expression is disqualified since `atm(A3)` has no joint with with `atm(A1)` or `atm(A2)` (although it has one with `atm(A4)`). At first sight it seems incorrect that this technique thus also disqualifies connected expressions such as

```
atm(A1)∧atm(A2)∧atm(A3)∧bond(A1,A3,B1)
∧bond(A2,A3,B2)∧single(B1)∧single(B2)
```

since `atm(A2)` has no joint with `atm(A1)`. This expression, however, is equivalent to the non-discarded counterpart where `A1` and `A3` are mutually exchanged. In can be shown also generally that no connected proper feature will be discarded by this technique, as long as one adheres to the (MI-S) assumption, that is, only one primary predicate is declared.

Unfortunately, I cannot provide any efficient feature connectivity verification technique for the remaining, most general (MI-M) case (multiple input arity + multiple primary predicates). I do not believe that one can be implemented without solving the general intractable graph connectivity testing problem. The (MI-M) case would correspond to a problem where individuals would be structured by two or more different manners, and it would be required to mutually relate such multiple kinds of substructures. This fortunately does not seem to be a typical case in applied propositionalization.

2.4 Avoiding Multiple Equivalent Features

The reader has certainly noticed an evident deficiency of the feature set corresponding to the set of all solutions to the dual HORN-SAT problem. The set may contain classes of equivalent features, only differing in variable naming. Indeed, if $\perp$ contains eg. multiple occurrences of the `load/2` predicate, as necessary for large enough n (refer to Fig. 1), two distinct solutions to the corresponding HORN-SAT instance will represent for instance the following, equivalent features:

```
car(C)∧load(C,L1)∧triangle(L1)
car(C)∧load(C,L2)∧triangle(L2)
```

Such cases may however be remedied by an extension of the generated HORN-SAT instance in the following way. Let $P_{b_1}, P_{b_2}, \ldots P_{b_n}$ be the propositional variables corresponding the (up to) n roots of the branches stemming from an atom in $\perp$ (again, refer to Fig. 1 for quicker insight). I add the following polynomial number (again due to the bounded branching factor and depth) of Horn clauses

$$\neg P_{b_1} \vee P_{b_2} \;,\; \neg P_{b_2} \vee P_{b_3} \;, \ldots \; \neg P_{b_{n-1}} \vee P_{b_n} \tag{8}$$

In the continuing example, $P_{b_1}, P_{b_2}, \ldots P_{b_n}$ correspond to the multiple `hasLoad/2` atoms contained and the clauses above will guarantee that `load(C,L`$_{i+1}$`)` ($i \geq 1$) will appear in a feature only if it also includes `load(C,L`$_i$`)`. To see that this technique does not eliminate features that are not redundant, realize that in the $\perp$ graph representation all children vertices of a given parent vertex have the same (up to variable naming) descendant subgraph, so any feature containing `load(C,L`$_{i+1}$`)` and not `load(C,L`$_i$`)` has its equivalent containing `load(C,L`$_i$`)` and not `load(C,L`$_{i+1}$`)`.

Still, the technique just explained does guarantee syntactic uniqueness of every feature in the resulting set. A thorough discussion of redundancy elimination is out of the scope of this paper; I just note here that the residual redundancy, exemplified by the following two equivalent features

```
car(C)∧load(C,L1)∧load(C,L2)∧triangle(L1)∧box(L2)
car(C)∧load(C,L1)∧load(C,L2)∧triangle(L2)∧box(L1)
```

manifests itself as well in the standard backtrack-search based feature construction systems and in neither framework there seems to be an apparent syntactic redundancy removal method not resorting to the NP-complete subsumption check.

2.5 Extracting Correct Features from the Bottom Feature

I am now in the position to extract correct features from $\perp$ by finding a satisfying assignment to a polynomial-size set of propositional Horn clauses. Horn satisfiability was identified as a tractable problem as early as in the 1970's [10] and later, efficient algorithms have been designed [5] able to find a *maximal (minimal)* solution, that is, one that assigns the *true* value to the greatest (smallest) possible number of variables, or determine that no solution exists. In this paper's context, a maximal solution corresponds to the smallest connected proper subexpression of $\perp$ (remind that a $\perp$'s atom belongs to the extracted subexpression if its corresponding propositional variable is false). Consequently, if the efficiently found maximal solution makes false n or fewer variables, I have found a correct feature. Otherwise, I can conclude that the declaration allows for no correct feature. In the continuing example, a maximal solution to the clauses constructed above makes true P_3, P_4 and P_5 (the reader will check that all seven clauses 1 – 5 are indeed satisfied), thus P_1 and P_2 are false. This corresponds to the correct feature `car(C)∧hasRoof(C)`.

So far I have merely shown how to efficiently decide the *feature existence* problem by finding a correct feature if one exists. In practice though, one will need to enumerate the entire set of correct features. For this purpose, fortunately, one can accommodate the algorithm proposed in [3] able to produce the set of all HORN-SAT instance solutions by iterative executions of the core procedure for finding a single solution. The input clause set is at each call modified in a way guaranteeing that the successive solutions form the entire (lexicographically ordered) set of solutions to the original HORN-SAT instance. A favorable property of the algorithm is that the total number of calls to the core procedure is polynomial in (i) the total number of literals in the original clause set, (ii) the number of existing solutions, that is, the algorithm does not introduce an exponential complexity factor when upgrading a single solution finding onto finding of all solutions. I refer the reader to [3] for further details. By employing this algorithm (in the way described in Fig. 3) to find all solutions to the HORN-SAT instance

```
$EnumerationOfFeatures(\mathcal{D}, n)$ : Given a correct user predicate declaration $\mathcal{D}$ and
    a number $n \geq 0$, produces the set of all proper connected features of size $\leq n$,
    satisfying $\mathcal{D}$.
    1. Construct bottom feature $\bot = \bot(\mathcal{D}, n)$.
    2. Construct HORN-SAT instance $\mathcal{H}$ from $\bot$.
    3. $\mathcal{S} := AllModels(\mathcal{H})$.
    4. For all $s \in \mathcal{S}$ with at most $n$ false assignments, convert $s$ into the corresponding
       feature $f$ and output $f$.
```

Fig. 3. Enumeration of all correct features through the HORN-SAT reduction strategy. Steps 1 and 2 are detailed in Section 2. Step 3, ie. procedure *AllModels* implements [3]: it terminates in polynomial time if the number of all models is polynomial. A correct declaration $\mathcal{D}$ complies to assumptions described in Section 2 (finiteness, $\prec$ order on types) and one of the (SI) or (MI-S) assumptions.

corresponding to the feature construction problem instance, I do not conduct significant 'excess computation' (corresponding to exploring exponentially large search subspaces containing no solution in the case of standard backtrack feature construction approaches), and specifically, if the actual number of correct features is polynomial, they are all enumerated in polynomial time.

3 Sampling the Feature Space

For the case when the total number of correct features allowed by the user declaration is exponential in n (maximal feature size) and complete feature enumeration is intractable, I offer two algorithms for feature sampling. Both of them approach the task by sampling in the space of models for the dual HORN-SAT problem.

The first algorithm described in Fig. 4 simply generates random truth assignments with at most n false-valued variables (corresponding to the maximal feature size n) to the dual HORN-SAT instance and checks (through a linear time algorithm) if they are models. To obtain a uniform sample of features, care must be taken to generate the truth assignments equiprobably, given the maximal feature length (n) constraint. The corresponding technique is explained in the Figure. Due to the uniformity, this algorithm is also able to produce an unbiased estimate of the total number of existing features. The number of iterations (truth assignments made) in the algorithm is linear in s (the required sample size) and $1/p$, where p is the actual proportion of the number of all models to the number of all possible truth assignments, which grows exponentially in n. Therefore, if the number of all correct features, ie. the number of all models of the dual problem is also exponential in n, the number of iterations is at most polynomial.

The second, locally deterministic algorithm shown in Fig. 6 can be viewed as a middle-ground between complete enumerative search and sampling. At each iteration it generates a random partition of the search space by assigning the true value to at least $|\mathcal{V}| - n$ randomly chosen variables in the dual HORN-SAT instance with $|\mathcal{V}|$ variables, thereby guaranteeing that any found model will convert to a feature of at most n atoms. The completion to a total truth

$SampleFeatures(\mathcal{D}, n, s)$: Given a correct user predicate declaration $\mathcal{D}$ and a number $n \geq 0$, produces a set $\mathcal{F}$ of s random features of size $\leq n$, satisfying $\mathcal{D}$ and an estimate e of the total number of such features.

```
1. Construct bottom feature $\perp = \perp(\mathcal{D}, n)$.
2. $\mathcal{H} = ConvertToHornSAT(\perp, n)$; $\mathcal{V}$ := the set of propositional variables in $\mathcal{H}$.
3. $t := 0; f := 0; \mathcal{F} := \{\}$
4. $\mathcal{T} := RandomTruthAssignment(\mathcal{V}, n)$; $t := t + 1$
5. if $ModelCheck(\mathcal{H}, \mathcal{T})$ then $\mathcal{F} := \mathcal{F} \cup \{ConvertToFeature(\perp, \mathcal{T})\}$; $f := f + 1$
6. if $f = s$ then return $\mathcal{F}$ and $e = f/t * \sum_{i=0}^{n} \binom{|\mathcal{V}|}{i}$, else go to 4
```

Fig. 4. Sampling features by generating random truth assignments to the dual HORN-SAT problem and checking whether they are models thereof. Procedures *ConvertToHornSAT* and *ConvertToFeature* implement reduction principles described in Section 2. Procedure *ModelCheck* implements a linear time HORN-SAT model checking algorithm. Proc. *RandomTruthAssignment* is described in Fig. 5.

$RandomTruthAssignment(\mathcal{V}, n)$: Given a set of propositional variables $\mathcal{V}$ and a number $n \leq |\mathcal{V}|$, produces a random truth assignment to $\mathcal{V}$ with at most n false assignments, with equal probability among all such assignments.

```
1. Choose a random number $0 \leq r \leq n$ with probability

   $$P(r) = \frac{\binom{|\mathcal{V}|}{r}}{\sum_{i=0}^{n} \binom{|\mathcal{V}|}{i}} \qquad (9)$$

2. Choose a random combination $C^r$ of $r$ variables from $\mathcal{V}$ with equal probability
   among all such combinations.
3. Output the assignment $\{false \leftarrow v_i | v_i \in C^r\} \cup \{v_j \leftarrow true | v_j \notin C^r\}$.
```

Fig. 5. Procedure *RandomTruthAssignment* ensures equiprobability by first selecting the number $r \leq n$ of false valued variables in the selected assignment with probability proportional to the number of all assignments with r false valued variables, and then drawing a random combination of r variables to be falsified

assignment is then done by formally adding the instantiated variables as positive singletons to the Horn clause set and then using a simple linear time-algorithm [5] for finding a minimal HORN-SAT model (instantiating the rest of the variables). Unlike the previous sampling algorithm, this locally deterministic algorithm does not guarantee that resulting features form a sample from a uniform distribution on all correct features, and thus it cannot provide an unbiased estimate of the total number of correct clauses. The non-uniformity is a consequence of both the bias toward the minimal model in each search space partition as well as the possible overlaps between individual partitions.

Although I formally use the name *ModelCheck* in Fig. 4 and *HornModel* in Fig. 6, I implemented the two procedures naturally by a single binary Prolog predicate, where the model carrying argument may or may not be instantiated

LocalSearchOfFeatures$(\mathcal{D}, n, s)$: Given a correct user predicate declaration $\mathcal{D}$ and a number $n \geq 0$, produces a set $\mathcal{F}$ of s random features of size $\leq n$, satisfying $\mathcal{D}$.

```
1. Construct bottom feature $\bot = \bot(\mathcal{D}, n)$.
2. $\mathcal{H} = ConvertToHornSAT(\bot, n)$; $\mathcal{V}$ := the set of propositional variables in $\mathcal{H}$.
3. $f := 0; \mathcal{F} := \{\}$
4. Choose a random number $|\mathcal{V}| - n \leq r \leq |\mathcal{V}|$ with probability
       $$P(r) = \frac{\binom{|\mathcal{V}|}{r}}{\sum_{i=|\mathcal{V}|-n}^{|\mathcal{V}|} \binom{|\mathcal{V}|}{i}} \tag{10}$$
5. Choose a random combination $C^r$ of $r$ variables from $\mathcal{V}$ with equal probability
   among all such combinations.
6. $\mathcal{H}^{ext} := \mathcal{H} \cup \{v_i \leftarrow true | v_i \in C^r\}$
7. if $\mathcal{T} := HornModel(\mathcal{H}^{ext})$ succeeds then
   $\mathcal{F} := \mathcal{F} \cup \{ConvertToFeature(\bot, \mathcal{T})\}$; $f := f + 1$
8. if $f = s$ then return $\mathcal{F}$, else go to 4
```

Fig. 6. Sampling features by generating a random, partial truth assignment to variables in the dual HORN-SAT theory, and then verifying if the assignment can be completed to a model of the theory. Procedure *HornModel* terminates in linear time [5].

when calling the predicate. The total number of calls to this predicate will represent a parameter used in comparing the two methods.

4 Implementation and Experiments

The algorithms presented in this paper have been implemented in SWI Prolog. The implementation is available for download from `http://labe.felk.cvut.cz/~zelezny/feature_sampling.pl` .

Let me now consider the following mode/type declaration

```
car(-c), connected(+c, +c), load(+c,-l), big(+l), small(+l)
```

and set the maximum branching factor $\beta = 2$ and maximum feature size $n = 10$. The bottom feature for this declaration has 18 atoms, and consequently there exist $\sum_{i=0}^{10} \binom{18}{i} = 199140$ possible truth assignments in the dual HORN-SAT instance (out of which 567 are models to the corresponding Horn theory).

I have two goals in this exercise. First, I want to verify that the estimate e of the total number of correct features provided by the algorithm *SampleFeatures* in Fig. 4 converges sufficiently rapidly to the correct value with growing sample size, in comparison to an estimate based on enumerating subsets of the bottom feature in systematic manners, either top-down or bottom-up. Second, I want to compare the efficiency of algorithms *SampleFeatures* in Fig. 4 and *LocalSearchOfFeatures* in Fig. 6 to see whether abandoning the distributional uniformity in *LocalSearchOfFeatures* trades off for a significant speedup of the feature sample construction, with respect to *SampleFeatures*.

Figure 7 answers the first question by demonstrating that the uniform sampling method (unlike the systematic top-down or bottom-up procedures) pro-

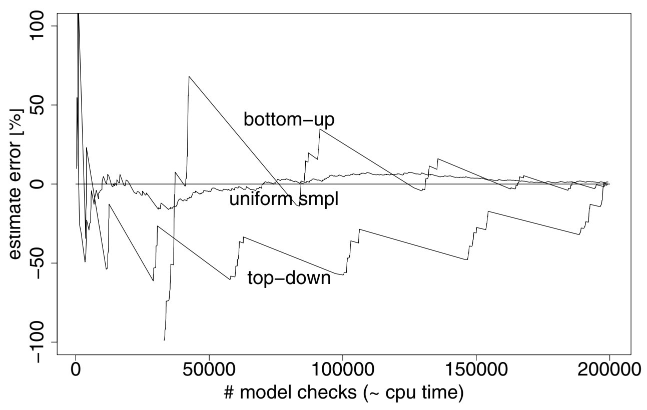

Fig. 7. The diagram plots the per cent error of the estimate e of the total number of correct features provided by the uniform sample in the *SampleFeatures* algorithm and by measuring the proportion of models of the dual HORN-SAT theory among possible truth assignments in a systematic enumeration of assignments starting either with n false valued variables (so that largest features are found first, denoted as bottom-up in the figure), or with 0 false valued variables (so that smallest features are found first, denoted as top-down in the figure). The error value is plotted against the growing number of verified truth assignments to the dual HORN-SAT problem.

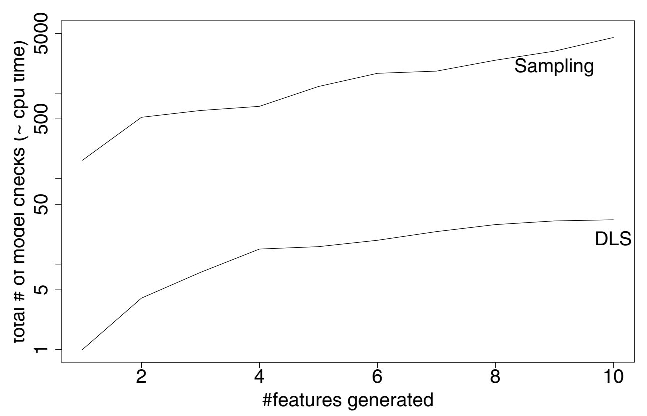

Fig. 8. The diagram plots the total number of calls to the *ModelCheck* (*HornModel*, respectively) procedure made in a pure sampling approach by the Algorithms *SampleFeature* (denoted as Sampling) and in a deterministic local search (denoted as DLS) by the *LocalSearch* algorithm. The value corresponds to the cpu time spent and is plotted on a logarithmic scale, against the growing number of features generated.

vides a stable estimate, which remains in a 15% error margin once about 5,000 truth assignments (about 2.5% of the search space) are sampled.

The second question is addressed in Fig. 8, showing that the rate at which the algorithm runtime increases with the growing number of feature sampled is reduced by $LocalSearchOfFeatures$ to about 1/100 of that invested by $SampleFeatures$.

5 Related Work

Similarly to the work of Pfahringer and Holmes [9], my approach aims at generating randomized features, viewable as extracting some constrained random subgraphs of some graph G. While in my approach, G (the bottom feature) is derived from a user's syntactic declaration, in [9], G is a structural representation of a chosen example of a class for which the features are generated. This difference has two fundamental consequences. First, unlike [9], my approach is class-blind and as such it is not apriori biased towards constructing features with discriminative power, which needs to be assessed (and possibly used for posterior feature selection) after the feature syntax has been generated. Second, Pfahringer's and Holmes' features are, to my best understanding, less expressive than my features. Assuming ground descriptions of examples, their feature graphs correspond to ground logic formulas, so unlike in my approach, they eg. cannot express the feature 'there are two carbon atoms c_1, c_2 both connected to some other atom A.

My method includes a number of ingredients, whose application in relational machine learning is not original. Namely, the concept of a bottom feature derived on the basis of user moding and typing declarations deliberately adopts the ideas of the mode directed inverse entailment [8] technique popular in ILP, where a *bottom clause* is constructed to constrain the search space. To my best knowledge though, there is no previous approach exploiting the 'bottom' concept for feature construction. The crucial point where I diverge from the traditional approach is in the utilization of the bottom feature: rather than using it as a constraint for a backtrack search, I translate it into a propositional Horn theory whose models represent correct features.

Furthermore, the basic idea of estimating the number of features by sampling in the feature space was inspired by an analogous technique implemented by Ashwin Srinivasan in the ILP system Aleph for estimating the number of existing legal clauses (see eg. [14]). Again, the principal difference of my materialization of that idea from the one in Aleph stems from the conversion of feature construction onto a propositional satisfiability problem enabling to carry out the sampling in the 'easily conquerable' space of propositional truth assignments.

6 Conclusion, Future Work

I have presented an approach to relational feature construction based on its poly-time reduction to the tractable HORN-SAT problem. Under acceptable restrictions on user language declarations, I am able to either efficiently enumerate the complete set of proper, connected, declaration compliant relational features (if their total number is polynomial in the maximum feature size), or otherwise efficiently obtain their random sample from the uniform distribution on the feature space.

The assumptions I imposed on the user declarations to achieve the results were its finite size, existing partial irreflexive order on types, which 'agrees' with the input-output order of types in any declared predicate (see Section 2 for details), and the 'non (MI-M)' condition which stipulates that either no two different declared predicates are primary (without input arguments), or no declared predicate has more than one input. Dropping the 'non (MI-M)' condition has the consequence that solutions arising from the dual HORN-SAT problem may correspond to disconnected features, which may be decomposed into two or more correct features and therefore are redundant.

Intuition suggests that the efficient technique used for solving the dual, satisfiability problem in linear time without backtracking, should have its 'mirror' procedure applicable directly on the primary problem of correct feature search. I have not yet been able to exactly determine what form such search procedure would acquire and this is a goal of my future work.

The approach admittedly needs more experimental evaluation. Also, the presented method for estimating the number of correct features (or, equivalently, their relative frequency $0 \leq p \leq 1$ in the search space) calls for a statistical analysis to determine the required size of the sample which, with a given probability, leads to an estimate $\hat{p}$ of p in a given error bound. Altough simple statistical techniques are available for calculating the error bound, they assume p not too close to 0 or 1. Unfortunately, p is typically very close to zero in the expression spaces in question. An alternative way, suggested by a reviewer, is to approach the estimate-reliability analysis empirically, eg. by repeated sampling.

Acknowledgement. Thanks go to the ILP 2005 reviewers for their informative suggestions and particularly for pointing out the relevant paper [9]. I am supported by the Czech Ministry of Education through the project 1ET101210513 "Relational Machine Learning for Biomedical Data Analysis" in the "Information Society" programme.

References

1. J. Blatak and L. Popelinsky. Distributed mining maximal first-order patterns. In *Work in Progress track of Inductive Logic Programming, 14th Inf. Conf*, 2004.
2. V. S. Costa, A. Srinivasan, R. Camacho, H. Blockeel, B.Demoen, G. Janssens, J. Struyf, H. Vandecasteele, and W.V. Laer. Query transformations for improving the efficiency of ilp systems. *J. Mach. Learn. Res.*, 4:465–491, 2003.
3. R. Dechter and A. Itai. Finding all solutions if you can find one. In *AAAI-92 Workshop on Tractable Reasoning*, 1992.
4. P. Domingos. Process-oriented estimation of generalization error. In *IJCAI97*, pages 714–721, 1999.

5. W. F. Dowling and J. H. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formula. *Journal of Logic Programming*, 3:267–284, 1994.
6. M-A. Krogel, S. Rawles, F. Železný, S. Wrobel, P. Flach, and N. Lavrac. Comparative evaluation of approaches to propositionalization. In *Proceedings of the 13th International Conference on Inductive Logic Programming*. Springer-Verlag, 2003.
7. N. Lavrač and P. A. Flach. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, 2(4):458–494, October 2001.
8. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
9. B. Pfahringer and G. Holmes. Propositionalization through stochastic discrimination. In *Work in Progress Track at Inductive Logic Programming, 13th Inf. Conf*, 2003.
10. T. J. Schaefer. The complexity of satisfiability problems. In *Tenth Annual Symposium on Theory of Computing*, pages 216–226, 1978.
11. M. Sebag and C. Rouveirol. Tractable induction and classification in first-order logic via stochastic matching. pages 888–893, 1997.
12. A. Srinivasan, S. H. Muggleton, M. J. E. Sternberg, and R. D. King. Theories for mutagenicity: a study in first-order and feature-based induction. *Artif. Intell.*, 85(1-2):277–299, 1996.
13. C. Vens, A.Van Assche, H. Blockeel, and Saso Dzeroski. First order random forests with complex aggregates. In *ILP*, pages 323–340, 2004.
14. F. Železný, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. volume 2583, pages 333–345, 2003.