2009 Ninth International Conference on Intelligent Systems Design and Applications

Grammatical Concept Representation for Randomised Optimisation Algorithms in Relational Learning

Petr Buryan, Jiří Kubalík
Department of Cybernetics,
Czech Technical University in Prague
Prague, Czech Republic
{buryan, kubalik}@labe.felk.cvut.cz

Katsumi Inoue
National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku,
Tokyo, Japan
ki@nii.ac.jp

Abstract— **This paper proposes a novel grammar-based framework of concept representation for randomized search in Relational Learning (RL), namely for Inductive Logic Programming. The utilization of grammars guarantees that the search operations produce syntactically correct concepts and that the background knowledge encoded in the grammar can be used both for directing the search and for restricting the space of possible concepts to relevant candidate concepts (semantically valid concepts). Not only that it enables handling and incorporating the domain knowledge in a declarative fashion, but grammars also make the new approach transparent, flexible, less problem-specific and allow it to be easily used by almost any randomized algorithm within RL. Initial test results suggest that the grammar-based algorithm has strong potential for RL tasks.**

Keywords: ILP; randomised search; grammars

I. Introduction

One of busy areas of Relational Learning (RL) [3] is Inductive Concept Learning (ICL) where a limited set of positive and negative examples called training set is used to induce a concept description. Challenging class of learning problems is the subfield of known as Inductive Logic Programming [1] (ILP). ILP uses a fragment of First Order Logic (FOL) as the hypothesis language and its fundamental characteristic is the ability to accommodate background knowledge in order to efficiently guide the search.

In this paper, we address the problem of relational domain representation that would allow random based optimization algorithms such as Evolutionary algorithms [4] to efficiently and effectively search for concepts in ILP. The specific characteristics of the relational domain (enormous size of the space of possible concepts with sparse presence of meaningful ones, necessity to incorporate background knowledge) still inhibit wider utilization of stochastic search in ILP.

We introduce a novel approach based on utilization of grammars which is inspired by Grammatical Evolution [5]. This technique is suitable for randomized optimization algorithms not only in ILP but also in other RL domains such as e.g. Graph Mining. Its main advantage is that it efficiently restricts the space of possible concepts towards relevant candidate concepts (semantically valid concepts). It also enables to handle and incorporate the domain knowledge in a transparent and declarative fashion. In addition, this approach is not limited to description of either trees only (Genetic Programming) or to utilisation of sampling based on propositionalisation.

The paper is organized as follows. Section 2 gives some necessary background and related work, Section 3 discusses our approach and Section 4 gives results for simple example experiment. Finally, Section 5 concludes and discusses some ideas for future research.

II. Background

In ILP, the problem domain is described in FOL which provides formal framework for concept description and any further reasoning.

Definition 1. *The basic setting of ILP [1]. Given a problem defined as triplet (B, E^+, E^-), where B is background knowledge, E^+ and E^- are sets of positive and negative examples, find a theory T, such that*

1. $e_p \in E^+$: $B \wedge T \models e_p$ (T is complete)

2. $e_n \in E^-$: $B \wedge T \not\models e_n$ (T is consistent)

The process of search for concepts can be seen as a search in space of candidate concepts. Starting from an initial hypothesis (theory), generalization and specialization operators are applied to direct the search towards good hypotheses that cover many positive examples and minimum of the negative ones.

A. ILP as Optimisation Problem

The search for the clause with optimal coverage in ILP is in principle just another example of optimization problem that is in case of ILP NP-complete [16]. The optimality of target hypothesis or concept may be judged by several optimization criteria, most often by its completeness, consistency, and simplicity [1]. Due to the search complexity, ILP systems have large time and storage requirements and therefore need efficient search strategies [13] (including randomized search).

Standard ILP systems (including the most popular systems FOIL [3], Aleph [2] and Progol [5]) in the search strategy employ approaches of sequential covering and hill climbing. The disadvantage of combination of greedy and hill climbing approach is its susceptibility to local optima.

Note that some conjunctions when taken alone may exhibit inferior quality but when considered together with other literals may constitute an optimal theory. One way to avoid getting trapped in local optima and increase the chance of converging to more fit solutions is to introduce a stochastic component into the search process.

Randomized algorithms have long been under focus of ILP research. To reach interesting concepts in shorter time we are willing to give up the completeness of the search in favour for quickly learning "sufficient" concept. Different basic randomized search algorithms have been developed and studied e.g. in works [15] and [18].In paper [18], stochastic search for bottom-up rule learning is compared to basic GSAT and WalkSAT.

Another approach to avoid being trapped in local optima is utilization of Evolutionary algorithms (EAs) [4]. EAs have been in the past successfully used for various concept learning problems including various systems for ILP [21]. In many evolutionary concept learners, the EAs just replace the hill climbing method for finding the best rule to be added into the emerging target concept. This is the case of systems ECL [6], REGAL [7], G-NET [14] or DOGMA [12]. Binary string representation (for which standard blind recombination operators can be used) is used for encoding the evolved hypotheses in all of them. Unfortunately, very sophisticated procedures are required for transforming hypotheses into these binary strings. For example, the latest approach [8] uses specially constructed binding matrix to transform the problem into binary array representation.

Different representation is used by system GLPS [9] that is based on Genetic Programming (GP) and evolves a population of complete logic programs. The programs evolved are represented as a forest of AND-OR trees, each representing a clause, and a standard GP-like crossover operator is used to generate new candidate programs from selected parents. Another system EVIL_1 [10] also evolves a population of logic programs, each encoded as a tree structure. However, traditional genetic operators are replaced with Progol algorithm that is used for inducing new rules and the crossover operator is used here to exchange rules between programs.

The tree representation of concepts used in the latter systems is more flexible as it allows considering rules of variable form. Such representation also allows utilization of effective tree-based crossover operators, where rules, clauses, or just parts of clauses can be easily swapped between two hypotheses.

Simulated Annealing (SA) [17] represents another stochastic optimisation technique that can be used in ILP. Inspired by physical annealing of solids it is an often used alternative to evolutionary techniques. First complete utilisation of SA in ILP may be seen in [17] (although there are some efforts already e.g. in the Aleph system). This SA adaptation works with ILP clauses and defines their neighbourhood by means of generalisation and specialisation operators.

However, almost all of these conventional concept learning approaches do not care about minimising syntactic errors and involve much effort in formulating a suitable representation and search operators. In addition, this formulation must be usually re-addressed for each problem domain thus making the systems hard-bound to certain problem domain. Generally, most of the above mentioned EA approaches use background knowledge only for the purposes of generating the initial population (if they use it at all). As a result, though bound to special type of concepts described (e.g. trees only in GP), the majority of randomised approaches to ILP fail to search only syntactically correct hypotheses and in consequence perform the search in too large space [11].

III. Grammars for ILP

All standard randomized optimization algorithms (except GP based GAs) work in their basic setting with linear data representation (usually with numeric arrays). Two key problems occurring here are: generation of meaningful clauses at random and their subsequent translation into the form of numeric arrays. Due to the enormous size of the space of potential hypotheses (infinite when variables are introduced), simple random clause generation has no effect as almost all such clauses cover either any or all of given examples. Efforts have been made to improve efficiency by incorporating background knowledge but these often require many constraints and are too problem-specific. We suggest a novel approach based on utilization of grammars to describe the problem domain and the structure of possible concepts.

Formal grammars, first introduced by Chomsky [19], are utilised in Computer Science to formalize the syntactic rules that are used to build or analyze phrases of given language. Inspired by GE [5], we use grammars in a generative approach: driven by numbers of numeric arrays that represent potential solutions, grammars are used to translate these arrays into logical clauses. Suitable form of such grammars comes out of context-free grammar defined in Backus-Naur form:

> ***Definition 2.*** *Grammar in Backus Naur Form (BNF). The BNF grammar is represented by the tuple {N, T, P, S}, where N is the set of non-terminals, T the set of terminals, S is a start symbol which is a member of N and P is a set of production rules* $r: N \rightarrow \{N \cup T\}^n$, *n=1, 2, ...*

The rules from *P* define how a symbol from *N* on the left side of rule is expanded into an ordered set of symbols of *N* and *T* on the right side of the rule. The final clause generated by the grammar consists only of elements of the terminal set *T*. This representation easily enables to implement all standard FOL features (such as negations, various logical conjunctions etc.) and can also easily go beyond FOL (e.g. utilization of numeric constraints). In addition, we suggest that each rule may be assigned numeric weight based either on the input data or on feedback from the user thereby adapting the approach of probabilistic grammars [22]. This offers an additional degree of freedom and another possibility to give the search desired bias.

By utilisation of grammars there is actually no need to develop specialised problem-specific data structures and/or new complicated search operators. It also requires minimal search bias (e.g. there is no necessary limit on clause length or on number of literals used).

A. Context Sensitive Grammars

Due to the enormous size of the hypotheses space it is impossible to rely on the fact that necessary relations shall sometime "just appear" during the search. It is important to ensure that each literal of the constructed clause is properly related to the rest of the clause and that such relations are not destroyed later by the search mechanism. Therefore, we suggest utilisation of context-sensitive grammar with the aim to define and efficiently restrict the search space and to smoothly incorporate the background knowledge.

The grammar has the form similar to BNF with that difference that each non-terminal symbol may be assigned a set of context constraints under which it can be later replaced (see Fig.3 for an example).

> ***Definition 3.*** *The context sensitive grammar for ILP we define as the tuple {N, T, P, S, C}. N is the set of non-terminals, T the set of terminals, S a start symbol, C is set of context related constraints and rules r from P take the form r:* $N \times C \rightarrow \{(N \times C) \cup T\}^n$*, n=1, 2, ..*

Except for limiting unwanted or unnecessary literal co-occurrence, the context-related constraints from C are important mainly for utilization of variables that define relations between literals (according to current state of the translated clause constraints can be used to restrict the utilization of variables within the argument of some specific literal only to those variables that are already used elsewhere before in the clause).

The pseudocode of basic constraint-defined context implementation into the translation process is given in Algorithm 1. Such implementation enables transparent context definition and even utilization of several constraints at once. Another advantage of context defined in the form of constraints is that these constraints can be automatically induced from given relational dataset before the start of the search process.

B. Grammar in Mapping Process

The numeric array representing the solution is translated into logical clause by following algorithm (see Algorithm 1 for pseudocode). Given an array of numbers *A* that shall be translated the algorithm proceeds with the translation from left to right of the array each time processing one number *N* from *A* and one non-terminal symbol *NT* from the clause.

At each step, initially a set of all rules that are generally applicable to *NT* is selected from the grammar. This set is then filtered according to context constraints bound to *NT* (each non-terminal *NT* carries its own specific set of context constraints). Finally, the number *N* is used to select one rule from this filtered set of rules (rule weights are also considered here) and the rule is applied to the clause.

Application of the rule results in replacing the non-terminal *NT* by set of symbols given by the right hand side of the production rule selected. These steps are repeated until the final clause is built (i.e. no non-terminals are present) or end of array *A* is reached. In case when end of array *A* is reached and some non-terminals still remain in the clause we either clear the remaining non-terminals from the clause or, in case when deletion is not possible (e.g. argument of literal), we replace them with anonymous variable (`_VAR`).

The translation is purely deterministic, each time the same array of numbers is translated it will always generate the same clause (though expression of each individual number within the array is not universal and depends on the current non-terminal to which it is being applied and the respective context).

Algorithm 1 Translation algorithm

```
Input: Array a, Grammar G
Output: Clause C
C = {start_symbol}
C_contains_nonterminals = true
pos_a = 1
while(C_contains_nonterminals) do
if (pos_a>length(a)) break while
N = number at position pos_a in a
NT = first non-terminal symbol from the left in C
RHS = get set of all production rules applicable for NT
          from grammar G
RHS' = filter out unsuitable rules from RHS acc. to
          context constraints bound with NT
R_N = N modulo size(RHS')
PR = select rule at position R_N from RHS'
C = replace NT in C according to the rule PR
if (still some non-terminals exist in C)
    pos_a++
 else
    C_contains_nonterminals=false
end while
if (C_contains_nonterminals) post-process(C)
return C
```

C. Grammar Induction

Inducing the context-sensitive grammar from given data represents a novel problem as we need to induce this grammar for a special purpose of creating hypotheses from an array of numbers. Here we present our first approach to solving it. The grammar should basically fulfil two basic requirements: it has to allow generation of all clauses that have some support in the data (a threshold may be applied here) and it should restrict the generation towards such clauses that have at least some coverage in the data.

The induction algorithm constructs basic context-sensitive grammar in the BNF-like form based on the input from the user that defines the target predicate, other predicates with their arity (number of arguments, in further text we use the notation 'predicate/arity') and constants to be used in the logical concept search. We start by constructing basic graph of relations G_R from the given dataset that describes the basic structure of the literals and their relations used in the dataset.

> ***Definition 4.*** *Graph of relations* G_R*.* $G_R = <V,E>$*, where V is the set of nodes each node of which corresponds to one argument of a literal or one constant and E is set of*

edges. Two nodes n_1, n_2 are connected by an edge from E if

- *n_1 corresponds to i-th argument of literal L, n_2 is constant C, C can be used as the i-th argument of L (such situation appears in given data); example query: '`odd(2)?`';*
- *n_1 corresponds to i-th argument of literal L_1, n_2 corresponds to j-th argument of literal L_2, the same variable can be used concurrently in both arguments n_1 and n_2 (such situation appears in given data); example queries: '`successor(X, X)?`' or '`even(X) and odd(X)?`'.*

After obtaining G_R we order its nodes hierarchically into layers based on the relations of predicates (the edges): first layer consists only of the target predicate nodes, the next layer consists of nodes of those predicates that are related directly with the target, third layer of predicates related directly with predicates from second layer (and not the first) and so on.

The grammar is built form G_R in following way: the first rule of the new grammar replaces initial start symbol by a conjunction of target predicate (first layer) and one non-terminal NT_0. Second rule defines the replacement of NT_0 by other terminal symbols from the second layer in conjunction with non-terminals that can be replaced by symbols from the same (second) and the next (third) layer. This principle is shared by further rules according to the relations that are discovered in the data. In addition, a set of context constraints is linked to each of the non-terminal symbols based on existing and non-existing edges in the graph G_R.

The context generally can include all predicates of the whole translated clause. We consider only basic graph of relations i.e. relation only of pairs of predicates when creating the grammar as this is the basic case for relational domains. Broader context scope may be considered but this implies that the number of queries that need to be performed during grammar induction as well as the time of translation would rise very fast.

Considering simple numeric dataset from Fig 1, Fig.2 shows an example of its graph G_R with only three predicates (`even/1`, `odd/1`, `successor/2`) and one constant (number '`2`') used. The graph G_R is built in order to be finally used to create the grammar for the search of the concept of even numbers `even/1`. An example of grammar that is induced from the graph is in Figure 3. In the grammar, the number of variables was limited to 2, the only constant available is '`2`', symbol '|' is used as delimiter in those cases, where there is more than one production rule applicable to one non-terminal.

```
even(2). even(4). even(6)....
 odd(1). odd(3). odd(5). ...

 successor(X,Y) :- X = Y + 1.
 successor(X,Y) :- Y = X - 1.
```

Figure 1. Simple even-odd dataset (in Prolog format)

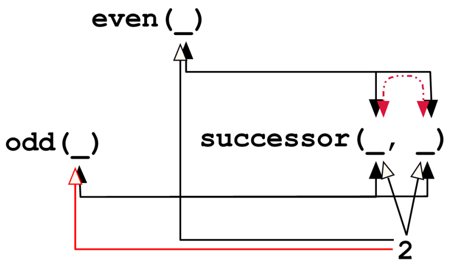

Figure 2. Graph of basic dataset structure (even/odd numbers, 3 literals, 1 constant utilisable). Full arrows represent potential relation between arguments, dashed line used for intra-argument relation, empty arrows stand for utilisation of constants. Black arrows signalise existing relation, red arrows signalize that there is no relation

```
(I) <start> → <lit0> ⇒ even(<var>)
(II) <lit0> → <lit0> and <lit0>   | (1)
            → odd(<lit1>+ctx1)     | (2)
            → successor(<lit1>+ctx2,
                        <lit1>+ctx3) (3)
(III) <lit1> → <var> |    (1)
             → 2          (2)
(IV) <var> → X1 | (1)
           → X2   (2)
```

Contexts:

ctx1: `no other replacement of <var> than the one already used in argument of successor/2 (any of both arguments)`

ctx2: `use replacement of <var> that is already used in argument of even/1 or odd/1`

ctx3: `use replacement of <var> that is already used in argument of even/1 or odd/1 or a new <var> (not used elsewhere), do not use <var> that is already used in first (sibling) argument of the very same predicate successor/2`

Figure 3. Grammar for search for the concept describing even numbers induced from graph in Fig.2

IV. EXPERIMENT

A short experiment is presented in this section using the basic *Trains Going East or Going West* problem proposed in [20] and its extension to randomly generated 1000 trains as presented in [8]. This dataset represents standard artificial learning problem in FOL and was primarily constructed to illustrate ILP learning capabilities. The dataset of 1000 trains was randomly generated by stochastic generator for the purposes of testing the abilities of grammars. Therefore, no hidden concept was used for discrimination between eastbound and westbound trains.

In our first experiment we analysed the ability of our approach to generate "useful" clauses i.e. such clauses that cover part of examples but not all of them. This is important not only for genesis of the initial population of clauses for evolutionary search but also for the whole function of the search, because modification of each member (clause) should ideally result back in "useful" clause.

TABLE I. COVERAGE OF 10.000 RANDOMLY GENERATED CLAUSES

Dataset	10 trains (original)			1000 trains (random)		
Avg. example size [literals in conj.]	21.3			47.3		
Array length [bytes]	16	32	48	16	32	48
Avg. clause length [literals in conj.]	3.7	7.6	11.5	3.6	7.6	11.5
Clauses with zero coverage [%]	12	62	80	5	37	78
Clauses with full coverage [%]	8	2	0	24	4	0
Other clauses [%]	**80**	**36**	**20**	**71**	**59**	**22**

The results summed in the Table 1 show the potential of our grammar-based concept. The grammar induced generated even for "random" conjunction of 11 literals 20% clauses with non-zero coverage. This seems to be a good starting position for randomised optimisers such as GE or SA. With implementation of more sophisticated context and more sophisticated induction algorithm this rate should further improve.

The second table (see Tab.2) gives brief comparison of stochastic GSAT search (Aleph[2] implementation) with greedy concept generation method that uses grammar framework and builds the concept by iterative steps (at each step adding several literals to the concept conjunction).

For this experiment we selected two datasets (5 eastbound trains and 5 westbound trains each) from the 1000 trains dataset, so that the shortest clause explaining all positive and no negative example of the new datasets contains 5 resp. 7 literals (tested by exhaustive search in Aleph). Required coverage of the new searched concept was set to all positive and no negative examples (this could be theoretically also accomplished by disjunction of several clauses but this was not the target of this test). Results summarising 30 runs are given in table Tab.2.

TABLE II. COMPARISON OF GSAT AND GRAMMAR-BASED GREEDY ALGORITHM

Min. concept length	**Algorithm type**	**Avg. clauses constructed**	**Standard Deviation**	**Succ. Runs [%](1**
5	GSAT	4724.5	5139.1	100
	Grammars	3835.2	3172.0	95
7	GSAT	6668.0	4797.1	45
	Grammars	4140.9	2854.9	95

(1 Number of runs terminated before reaching limit of 20.000 clauses (avg. from 30 runs)

Apart from better consistency of stochastic grammar-based method (lower std. deviation) these results indicate better efficiency of grammar-based search especially in the dataset that contains larger concepts only.

V. CONCLUDING REMARKS AND FURTHER WORK

In this paper, we approached the problem of learning interesting concepts in relational domains (namely ILP) as an optimisation task with the focus on utilisation of randomised searchers. We proposed a novel approach to problem representation that is based on context-sensitive rewriting grammars. We also presented basic algorithm for inducing this grammar from analysed data. Our approach is not restricted to description of either trees-only (GP) neither to utilisation of sampling based on propositionalisation.

Grammars represent a novel and suitable alternative for representation of logical hypotheses as series of numbers. They showed they are able to handle and incorporate the domain knowledge in a transparent and declarative fashion and therefore are suitable for utilisation within randomised searchers (GAs, SA etc.) in the relational domains. They can also easily reach beyond the scope of FOL (e.g. numeric constraints).

Further work will focus mainly on following issues:

- thorough comparison with state of the art techniques (work in progress) on various datasets;
- development of proper schemes for fast array translation and clause coverage calculation;
- analysis of expressivity and detailed requirements on the grammars, utilization of non-rewriting symbols (directives) the context and its scope refinement;
- full implementation within a GE system.

ACKNOWLEDGMENT

This work was supported by Research programme of Czech Ministry of Education (No. 1M0567).

REFERENCES

[1] S.H. Muggleton. *Inductive Logic Programming.* New Generation Computing, 8(4):295-318, 1991.

[2] Srinivasan, A. and Camacho, R.: "The Aleph Manual", http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph, 1993.

[3] J. R. Quinlan, "Learning logical definitions from relations", *Machine Learning,* 5(3):239-266, 1990.

[4] D. E. Goldberg: *Genetic algorithms for search, optimization, and machine learning*. Addison-Wesley, 1989.

[5] M. O'Neill and C. Ryan: Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers, 2002

[6] F. Divina and E. Marchiori, "Evolutionary concept learning", in *GECCO 2002: Proc. of the Genetic and Evolutionary Computation Conf.*, New York, Morgan Kaufmann Publishers, pp. 343-350, 2002.

[7] A. Giordana and F. Neri, "Search-intensive concept induction", *Evolutionary Computation Journal*, Vol. 3, pp. 375-416, 1996.

[8] A. Tamaddoni-Nezhad and S. Muggleton: "A Genetic Algorithms Approach to ILP", Proceedings of ILP 2002, *12th Intl. Conference on Inductive Logic Programming*, Sydney, Australia, 2002.

[9] M. L. Wong and K. S. Leung, "Inducing Logic Programs With Genetic Algorithms: The Genetic Logic Programming System", *in IEEE Expert* 10(5), pp. 68-76, 1995.

[10] P. Reiser: "EVIL1: A Learning System To Evolve Logical Theories", *In Proc. Workshop on Logic Programming and Multi-Agent Systems (International Conference on Logic Programming)*, pp. 28-34, 1997.

[11] P. Reiser: "Evolutionary Algorithms for Learning Formulae in First-order Logic", *dissertation thesis*, University of Wales, 1999.

[12] J. Hekanaho: "DOGMA: A GA-Based Relational Learner", *in Proceedings of Inductive Logic Programming, 8th International Workshop*, USA, 1998.

[13] W.W.Cohen: "Grammatically biased learning: learning logic programs using an explicit antecedent description language", *Artificial Intelligence*, 68(2), p.303-366, 1994.

[14] C. Anglano, A. Giordana, G. L. Bello, and L. Saitta: "An Experimental Evaluation Of Coevolutive Concept Learning", *in Proc. 15th Intl. Conf. on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pp. 19-27, 1998.

[15] F. Zelezny, A. Srinivasan, D. Page: "Randomized Restarted Search in ILP", *Machine Learning* 64(1-2): pp.183-208, 2006.

[16] E. Dantsin, T. Eiter, G. Gottlob and A. Voronkov. "Complexity And Expressive Power Of Logic Programming", *ACM Computing Surveys*, 33:374-425, 2001.

[17] M. Serrurier, H. Prade, and G. Richard: "A Simulated Annealing Framework for ILP", *In: R. Camacho, R. King, A. Srinivasan (Eds.): ILP 2004,* LNAI 3194, pp. 288–304, Springer-Verlag Berlin Heidelberg, 2004.

[18] U. Rückert & S. Kramer: "Stochastic Local Search In k-term DNF Learning", *Proc.of 20th Intl. Conf. on Machine Learning*, pp. 648-655, 2002.

[19] Chomsky, N.: *Syntactic Structures*. Mouton, 1957.

[20] R.S. Michalski: "Pattern Recognition As Rule-Guided Inductive Inference", *In Proc. IEEE Trans. on Pattern Analysis and Machine Intelligence*, p. 349–361, 1980.

[21] F. Divina: "Evolutionary Concept Learning in First Order Logic: An Overview". *AI Communications*. IOS Press, Volume 19, Number 1, pp. 13-33, 2006.

[22] Abney, S. P. "Stochastic Attribute-Value Grammars", *Computational Linguistics*, 23(4):597–617, 1997.