Approximate reasoning for efficient anytime induction from relational knowledge bases

Nicola Di Mauro and Teresa M.A. Basile and Stefano Ferilli and Floriana Esposito

Università degli Studi di Bari, Dipartimento di Informatica, 70125 Bari, Italy
{ndm,basile,ferilli,esposito}@di.uniba.it

Abstract. In most real-world applications the choice of the right representation language represents a fundamental issue, since it may give opportunities for generalization and make inductive reasoning computationally easier or harder. While the setting of First Order Logic (FOL) is the most suitable one to model the multi-relational data of real and complex domains, on the other hand it puts the question of the computational complexity of the knowledge induction that represents a challenge for multi-relational data mining algorithms. Indeed, the complexity of most real domains, in which a lot of relationships are required to model the objects involved, calls for both an efficient and effective search method for exploring the space of candidate solutions and a deduction procedure assessing the validity of the discovered knowledge. A way of tackling the complexity of such domains is to use a method that reformulates a multi-relational learning task into an attribute-value one. In this paper we propose an *approximate reasoning* technique that decreases the complexity of a relational problem changing both the language and the inference operation used for the deduction. The complexity of the FOL language is decreased by means of a stochastic propositionalization method, while the NP-completeness of the deduction is tackled using an *approximate query evaluation*. The induction is performed with an *anytime algorithm*, implemented by a population based method, able to efficiently extract knowledge from structured data in form of complete FOL definitions. The validity of the proposed technique has been proved making an empirical evaluation on a real-world dataset.

1 Motivations

Over the last decades large volumes of data in digital form have been acquired. Most of these data are stored using relational databases consisting of multiple tables and associations. Moreover, the data used in the fields of weather prediction, financial risk analysis and drug design are relational in nature. The induction of conceptual definitions to model the knowledge of such complex real-world domains is a hard and crucial task. The challenges posed by such domains are due to various elements such as the noise in the descriptions, the lack of data, but also the choice of the right representation language exploited to describe them.

Representation is a fundamental as well as a critical aspect in the process of knowledge discovery. Indeed, the choice of the right representation has a significant impact on the performance of the learning algorithms but also on the possibility to interpret and

reuse the discovered knowledge. The most suitable representation language to describe the objects and their relationships of complex real-world domains is a logic-based representation such as the first-order logic language (FOL) and the research area implementing algorithms working on this kind of representation is popularly known as Inductive Logic Programming (ILP). ILP systems represent examples, background knowledge, hypotheses and target concepts in Horn clause logic. The core of ILP is the use of logic for representation and the search for syntactically legal hypotheses constructed from predicates provided by the background knowledge.

However, this representation language allows a potentially large number of mappings between descriptions. The obvious consequence of such a representation is that both the space of candidate solutions to search and the test to assess the validity of the induced model result more costly. A possible solution is represented by *approximate reasoning* techniques [1], that try to decrease the complexity of a problem changing either the adopted language or the inference operation used for the deduction. In this way, the results may be unsound or incomplete but with a consequent speed-up and a reduced reasoning complexity.

A possible approximate reasoning technique consists in reformulating the original multi-relational learning task in a propositional one (i.e., *propositionalization*). This reformulation can be partial (heuristic), in which information is lost and the representation change is incomplete, or complete, in which no information is lost. In general, however, it is not possible to efficiently transform multi-relational data into an equivalent propositional form without an exponentially increasing complexity [2]. Alternative approaches concern the possibility to apply propositionalization directly on the original FOL context by a sort of flattening of the multi-relational data substituting them with all (or a subset of) their matchings with a pattern that can be provided by the users or previously built by the system.

This work proposes a method to decrease the dimensionality of the space of candidate solutions of multi-relational data to search by means of a propositionalization technique in which the transposition of the relational data is performed by an *online* flattening of the examples. The proposed method is a population based (genetic) algorithm that stochastically propositionalizes the training examples in which the learning phase may be viewed as a bottom-up search in the hypotheses space. The objective of this paper is twofold:

O1: providing an efficient and scalable method for inductive reasoning on relational databases, combining a genetic approach to navigate the search space of candidate solutions with a partial transposition of the relational knowledge base in a propositional one;

O2: incorporating an approximate reasoning strategy in a relational inductive learner, making incomplete a) the validation test (*query answering*) of the acquired knowledge, and b) the inductive generalization task.

The resulting learning algorithm, of the objective O1, belongs to the class of *anytime* algorithms [3] whose quality of results improves gradually as computation time increases, hence trading this quality against the cost of computation. They are resource constrained algorithms that return the best solution within a specified computational budget.

The validation test of the acquired knowledge, in objective O2, corresponds to the classical query answering task, that in relational learning is obtained by solving a subsumption problem known to be NP-complete. In many cases it is less important to obtain an exact query result than keeping query response time short. For instance, the following conjunctive FOL query

$$author(A1, P1), journal(P1, J), impact(J, IF), author(A2, P2), journal(P2, J)$$

may be used to test if there exist two authors $A1$ and $A2$ that have published a paper (resp. $P1$ and $P2$) in a journal J with an impact factor IF. Sometimes, instead of a correct answer, it may be suitable knowing the response for a subset of the authors in the domain, sacrificing the accuracy to improve running time. The approximate query answering used in this work is a *sampling* based technique, in which a random sample of the individuals involved in the domain is selected and used to solve the query.

2 Related Work

Various strategies have been proposed in order to overcome the limitation imposed by the inborn complexity of most real-word applications whose descriptions involve many relationships. One of the classical approaches consists in the reformulation of the relational learning in a propositional one followed by the application of well known propositional learners and by the mapping back in relational form of the resulting hypotheses. During the reformulation, a fixed set of structural features is built from relational background knowledge and the structural properties of the individuals occurring in the examples. In such a process, each feature is defined in terms of a corresponding program clause whose body is made up of a set of literals derived from the relational background knowledge. When the clause defining the feature is called for a particular individual (i.e., if its argument is bound to some example identifier) and this call succeeds at least once, the corresponding boolean feature is defined to be true for the given example; otherwise, it is defined to be false. Examples of systems that implement such kind of propositionalization process are LINUS [4], and its extensions DINUS [5] and SINUS [6], and RSD [7].

Alternative approaches avoid the reformulation process and apply propositionalization directly on the original FOL context: the relational examples are *flattened* by substituting them with all (or a subset of) their matchings with a pattern. To this concern, it was noted [8–10] that a most suitable setting for supervised relational learning is that of multiple instance problems (MIP), first introduced by Dietterich [11], where each example consists of a set of literals (instances) built on a same predicate symbol. The multiple instances representation is an extension that offers a good trade-off between the expressive power of relational learning and the low complexity of propositional learning. Unfortunately, most of the existing inductive learning systems are not able to face efficiently with this problem. In some cases the learning system has an incomplete knowledge about each training example: It does not know the features vector but it only knows that each example can be represented by means of one (or more) potential feature vectors (a bag of instances).

Following this idea, [12, 9, 10] proposed a multi-instance propositionalization. In such a framework each relational example is reformulated in its multiple matchings with a pattern (a formula of the initial hypotheses space that can be built from the training data or provided by the user). After the reformulation, each initial observation corresponds to many feature vectors and the search for hypotheses may be recasted in this propositional representation as the search for rules that cover at least one instance per observation. Consequently, the learning task is no longer to induce an hypothesis that is consistent with all the feature vectors reformulated but an hypothesis that covers at least one reformulated example of each positive initial training example and no reformulated example of any negative initial training example.

Specifically, the approach proposed in [9] consists in limiting the number of possible mappings by means of a selective mapping and then searching inductive generalizations in the hypotheses space defined by the selected mapping type. The type of mapping, i.e. the relevant propositionalization pattern, is provided by the user/expert and represents a (strong) bias which allows to dramatically reduce the matching space. On the contrary, in [12] the propositionalization process is done through a stochastic selection on each example of a user-defined number of example matchings with the pattern, which allows to reduce the dimensionality of the reformulated problem. In other words, for each example it is constructed the set of a user-defined number of hypotheses covering the example and not covering any example belonging to other classes. Then a representative of such a set for each example is learned that classifies unseen examples via a nearest-neighbour-like process. Finally, [10] proposed a method that selectively propositionalizes the relational data by interleaving attribute-value reformulation and algebraic resolution avoiding, as much as possible, the generation of reformulated data which are not relevant with respect to the discrimination task and obtaining a reformulated learning problem of tractable size. The obtained set of attribute-value instances is then used to solve the initial relational problem by applying a data-driven strategy.

Based on this kind of more suitable propositionalization and on the existing effective and efficient techniques for feature selection, [13] proposed an extension of classical feature selection methods for coping with the problem of relational data by firstly transforming the original relational data in propositional ones by means of a multi-instance propositionalization and successively applying methods for feature selection on such a new transformation.

3 The anytime induction method

In this paper we propose a technique that, reformulating the training positive and negative examples, solves the multi-relational learning problem by applying a data-driven bottom-up strategy.

3.1 Logic background

We used Datalog [14] as representation language for the domain and induced knowledge, that here is briefly reviewed. For a more comprehensive introduction to logic programming and ILP we refer the reader to [15, 16, 5].

A first-order *alphabet* consists of a set of *constants*, a set of *variables*, a set of *function symbols*, and a non-empty set of *predicate symbols*. Each function symbol and each predicate symbol has a natural number (its *arity*) assigned to it. The arity assigned to a function symbol represents the number of arguments the function has. Constants may be viewed as function symbols of arity 0. A *term* is a constant symbol, a variable symbols, or an n-ary function symbol f applied to n terms $t_1, t_2, \ldots, t_n$.

An atom $p(t_1, \ldots, t_n)$ (or atomic formula) is a predicate symbol p of arity n applied to n terms t_i. Both l and its negation $\bar{l}$ are said to be *literals* (resp. positive and negative literal) whenever l is an atomic formula. A *clause* is a formula of the form $\forall X_1 \forall X_2 \ldots \forall X_n (L_1 \vee L_2 \vee \ldots \vee \overline{L}_i \vee \overline{L}_{i+1} \vee \ldots \vee \overline{L}_m)$ where each L_i is a literal and $X_1, X_2, \ldots X_n$ are all the variables occurring in $L_1 \vee L_2 \vee \ldots \overline{L}_i \vee \ldots \overline{L}_m$. Most commonly the same clause is written as an implication $L_1, L_2, \ldots L_{i-1} \leftarrow L_i, L_{i+1}, \ldots L_m$, where $L_1, L_2, \ldots L_{i-1}$ is the *head* of the clause and $L_i, L_{i+1}, \ldots L_m$ is the *body* of the clause. Clauses, literals and terms are said to be *ground* whenever they do not contain variables. A *Horn clause* is a clause which contains at most one positive literal. A *Datalog clause* is a clause with no function symbols of non-zero arity; only variables and constants can be used as predicate arguments.

A *substitution* θ is defined as a set of bindings $\{X_1 \leftarrow a_1, \ldots, X_n \leftarrow a_n\}$ where $X_i, 1 \leq i \leq n$ is a variable and $a_i, 1 \leq i \leq n$ is a term. A substitution θ is applicable to an expression e, obtaining the expression $e\theta$, by replacing all variables X_i with their corresponding terms a_i.

The learning problem for ILP can be formally defined:

Given: A finite set of clauses $\mathcal{B}$ (*background knowledge*) and sets of clauses E^+ and E^- (positive and negative *examples*).

Find: A theory Σ (a finite set of clauses), such that $\Sigma \cup \mathcal{B}$ is *correct* with respect to E^+ and E^-, i.e.: a) $\Sigma \cup \mathcal{B}$ is *complete* with respect to E^+: $\Sigma \cup \mathcal{B} \models E^+$; and, b) $\Sigma \cup \mathcal{B}$ is *consistent* with respect to E^-: $\Sigma \cup \mathcal{B} \not\models E^-$.

Given the formula $\Sigma \cup \mathcal{B} \models E^+$, deriving E^+ from $\Sigma \cup \mathcal{B}$ is *deduction*, and deriving Σ from $\mathcal{B}$ and E^+ is *induction*. In the simplest model, $\mathcal{B}$ is supposed to be empty and the deductive inference rule $\models$ corresponds to θ-*subsumption* between clauses.

Definition 1 (θ-subsumption). *A clause c_1 θ-subsumes a clause c_2 if and only if there exists a substitution σ such that $c_1\sigma \subseteq c_2$. c_1 is a generalization of c_2 (and c_2 a specialization of c_1) under θ-subsumption. If c_1 θ-subsumes c_2 then $c_1 \models c_2$.*

θ-subsumption is the test used in relational learning for query answering. It corresponds to the most time consuming task of the induction process being it a problem NP-complete. In Section 3.4 we will present an approximate θ-subsumption test based on a sampling method described in the following section.

3.2 Data Reformulation

The method we propose is based on a stochastic reformulation of examples that, differently from other proposed propositionalization techniques, does not use the classical subsumption relation. For instance, in PROPAL [10], each example E, described in

FOL, is reformulated into a set of matchings of a propositional pattern P with E by using the classical θ-subsumption procedure, being in this way still bound to the FOL context. On the contrary, in our approach the reformulation is based on a *syntactic rewriting* of the training examples based on a fixed set of domain constants.

Let E be an example, represented as a Datalog ground clause, and let $consts(E)$ be the set of the constants appearing in E. One can write a new example E' from E by changing one or more constants in E, i.e. by renaming. In particular, E' may be obtained by applying an antisubstitution (i.e., a mapping from terms onto variables) and a substitution under Object Identity (OI) to E, $E' = E\sigma^{-1}\theta_{OI}$, where σ^{-1} is an antisubstitution that maps terms to variables, and θ_{OI} is a substitution under OI. In the Object Identity framework, within a clause, terms that are denoted with different symbols must be distinct, i.e. they must represent different objects of the domain. In the following we will omit the OI notation, and we will consider substitutions under the Object Identity framework.

Definition 2 (Renaming of an example). *A ground renaming of an example E, $R(E)$, is obtained by applying a substitution $\theta = \{V_1/t_1, V_2/t_2, \dots V_n/t_n\}$ to $E\sigma^{-1}$, i.e. $R(E) = E\sigma^{-1}\theta$, such that σ^{-1} is an antisubstitution, $\{V_1, V_2, \dots V_n\} \subseteq vars(E\sigma^{-1})$, and $\{t_1, t_2, \dots t_n\}$ are distinct constants of $consts(E)$, $n = consts(E)$.*

Example 1. Let $E : h(a) \leftarrow q(a,b), c(b), t(b,c)$ an example, $C = consts(E) = \{a, b, c\}$, and $\sigma^{-1} = \{a/X, b/Y, c/Z\}$ an antisubstitution. All the possible ground renamings of E, $\mathcal{R}(E)$ in the following, are

$E_1 : h(a) \leftarrow q(a,b), c(b), t(b,c)$,
$E_2 : h(a) \leftarrow q(a,c), c(c), t(c,b)$,
$E_3 : h(b) \leftarrow q(b,a), c(a), t(a,c)$,
$E_4 : h(b) \leftarrow q(b,c), c(c), t(c,a)$,
$E_5 : h(c) \leftarrow q(c,a), c(a), t(a,b)$,
$E_6 : h(c) \leftarrow q(c,b), c(b), t(b,a)$

obtained by applying to $E\sigma^{-1} : h(X) \leftarrow q(X,Y), c(Y), t(Y,Z)$ all the possible injective substitutions from $vars(E\sigma^{-1}) = \{X, Y, Z\}$ to $consts(E)$.

In this way, we do not need to use the θ-subsumption test to compute the renamings of an example E, we just have to rewrite it considering the permutations of the constants in $consts(E)$.

Lemma 1. *Given an example E, let $m = \mid consts(E) \mid$. The number of all possible renamings of E, $|\mathcal{R}(E)|$, is equal to the number of permutations on a set of m constants, i.e. $|\mathcal{R}(E)| = P_m^m = m!$.*

Proof. Let $consts(E) = \{c_1, c_2, \dots, c_m\}$, and $\sigma^{-1} = \{c_1/V_1, c_2/V_2, \dots, c_m/V_m\}$ be an antisubstitution. By Definition 2, a renaming $R(E) \in \mathcal{R}(E)$ is obtained by choosing a substitution $\theta_i = \{V_1/t_{1i}, V_2/t_{2i}, \dots, V_m/t_{mi}\}$, where $\{t_{1i}, t_{2i} \dots, t_{mi}\}$ are elements of $consts(E)$, s.t. $R(E) = E\sigma^{-1}\theta_i$. Letting fixed variables $V_j, j = 1 \dots m$, all the possible substitutions θ_i can be obtained by selecting permutations $(t_{1i}t_{2i}\cdots t_{mi})_i$ of the elements in the set $\{c_1, c_2, \dots, c_m\}$. Being $P_m^m = m!$, it follows that $|\mathcal{R}(E)| = |\{R(E) \mid R(E) = E\sigma^{-1}\theta_i\}| = P_m^m = m!$. ◁

Lemma 2. *All the renamings of an example E belong to the same equivalence class, $[E] = \mathcal{R}(E) = \{R(E) \in \mathcal{E} \mid R(E) \sim_s E\}$, based on the equivalence relation $\sim_s$ defined by $a \sim_s b$ iff a is syntactically equivalent to b, where $\mathcal{E}$ is the set of all the possible ground clauses. In particular, given an example E, $\forall E' \in [E], \exists \theta, \sigma^{-1}$ s.t. $E'\sigma^{-1}\theta = E$.*

Proof. Let $consts(E) = \{c_1, c_2, \ldots, c_m\}$. If $R, Q \in \mathcal{R}(E)$ then, by Definition 2, $\exists \sigma^{-1} = \{c_1/V_1, c_2/V_2, \ldots, c_m/V_m\}$, and $\theta_R = \{V_1/t_{1R}, \ldots, V_m/t_{mR}\}$ and $\theta_Q = \{V_1/t_{1Q}, \ldots, V_m/t_{mQ}\}$, where $(t_{1R} \cdots t_{mR})$ and $(t_{1Q} \cdots t_{mQ})$ are permutations of the elements in the set $consts(E)$, s.t. $R = E\sigma^{-1}\theta_R$ and $Q = E\sigma^{-1}\theta_Q$. Now, $R\theta_R^{-1}\sigma = Q\theta_Q^{-1}\sigma$, where $\theta_R^{-1} = \{t_{1R}/V_1, \ldots, t_{mR}/V_m\}$, $\theta_Q^{-1} = \{t_{1Q}/V_1, \ldots, t_{mQ}/V_m\}$ and $\sigma = \{V_1/c_1, \ldots, V_m/c_m\}$, and hence R and Q are syntactically equivalent, $R \sim_s Q$. ◁

Table 1 reports the propositional representation of the renamings belonging to the equivalence class of the clause reported in the Example 1.

	h(a)	h(b)	h(c)	q(a,b)	q(a,c)	q(b,a)	q(b,c)	q(c,a)	q(c,b)	c(a)	c(b)	c(c)	t(a,b)	t(a,c)	t(b,a)	t(b,c)	t(c,a)	t(c,b)
E_1	•			•							•					•		
E_2	•				•							•						•
E_3		•				•				•				•				
E_4		•					•					•					•	
E_5			•					•		•			•					•
E_6			•						•		•				•			

Table 1. Renamings of the clause $h(a) \leftarrow q(a, b), c(b), t(b, c)$

3.3 Approximate Model Construction

In the general framework of ILP, the generalization of clauses, and hence the model construction, is based on the concept of *least general generalization* originally introduced by Plotkin. Given two clauses C_1 and C_2, C_1 generalizes C_2 (denoted by $C_1 \leq C_2$) if C_1 subsumes C_2, i.e. there exists a substitution θ such that $C_1\theta \subseteq C_2$.

In our propositionalization framework, a generalization C (a non-ground clause) of two positive examples E_1 and E_2 may be calculated by turning constants into variables in the intersection between a renaming of E_1 and a renaming of E_2.

Definition 3. *Let E_1 and E_2 be two positive examples, n and m the number of constants in E_1 and E_2 respectively. Let C be a set of p constants such that $p \geq n$ and $p \geq m$. $R(E_1)_{\{C\}}$ and $R(E_2)_{\{C\}}$ indicate two generic renamings of the examples E_1 and E_2, respectively, onto the set of constants C.*

Proposition 1 (Generalization). *Given E_1, E_2 examples, a generalization G such that subsumes both E_1 and E_2, $G \leq E_1, E_2$ is*

$$G = (R(E_1)_{\{C\}} \cap R(E_2)_{\{C\}})\sigma^{-1}.$$

Proof. We must show, by generalization definition, that there exist θ_1, θ_2 substitutions, such that $G\theta_1 \subseteq E_1$ and $G\theta_2 \subseteq E_2$. $\forall l_j \in G\theta_i : l_j \in (R(E_1)_{\{C\}} \cap R(E_2)_{\{C\}})\sigma^{-1}\theta_i$, and hence $l_j \in R(E_i)_{\{C\}}\sigma^{-1}\theta_i$. θ_i are substitutions that map variables in G onto terms in E_i. Since $R(E_i)_{\{C\}}\sigma^{-1}\theta_i \in [E_i]$ then $R(E_i)_{\{C\}}\sigma^{-1}\theta_i \sim_s E_i$ by Lemma 2. Thus, $\forall l_j \in G\theta_i : l_j \in E_i$, hence $G\theta_i \subseteq E_i$. ◁

In order to obtain consistent intersections, it is important to note that all the renamings, for both E_1 and E_2, must be calculated on the same fixed set of constants. Hence, given $E_1, E_2, \ldots, E_n$ examples, the set C of the constants useful to build the renamings may be chosen equal to

$$C = \operatorname*{argmax}_{E_i}(|consts(E_i)|).$$

Furthermore, to avoid empty generalizations, the constants appearing in the head literal of the renamings must be take fixed.

Example 2. Given two positive examples
$E_1 : h(a) \leftarrow q(a,b), c(b), t(b,c), p(c,d)$ and
$E_2 : h(d) \leftarrow q(d,e), c(d), t(e,f)$.
We calculate C as:

$$C = \operatorname*{argmax}_{E_i}(|consts(E_i)|) = consts(E_1) = \{a, b, c, d\}.$$

Now,
$R(E_1)_{\{C\}} = \{h(a), \neg q(a,b), \neg c(b), \neg t(b,c), \neg p(c,d)\}$,
$R(E_2)_{\{C\}} = \{h(a), \neg q(a,b), \neg c(a), \neg t(b,c)\}$
A generalization G of E_1 and E_2 is
$G = (R(E_1)_{\{C\}} \cap R(E_2)_{\{C\}})\sigma^{-1} = \{h(a), \neg q(a,b), \neg t(b,c)\}\sigma^{-1} =$
$= (h(a) \leftarrow q(a,b), t(b,c))\sigma^{-1} = h(X) \leftarrow q(X,Y), t(Y,Z)$
with $\sigma^{-1} = \{a/X, b/Y, c/Z\}$.

3.4 Approximate Model Validation

The model validation we adopt in the proposed framework to assess and exploit the generated model on the seen and unseen data is based on a syntactic lazy matching.

Corollary 1 (Subsumption). *Given a generalization G and an example E, G subsumes E iff $R(G\theta)_{\{C\}} \cap R(E)_{\{C\}} \sim_s G\theta$.*

Proof. $\rightarrow$) If G subsumes E then, by definition, there exists a substitution θ s.t. $G\theta \subseteq E$. This means that $\forall l \in G\theta : l \in E$ and hence $G\theta \cap E = G\theta \sim_s R(G\theta)_{\{C\}} = R(G\theta \cap E)_{\{C\}} = R(G\theta)_{\{C\}} \cap R(E)_{\{C\}}$.
$\leftarrow$) If $R(G\theta)_{\{C\}} \cap R(E)_{\{C\}} \sim_s G\theta$, then by Proposition 1,
$(R(G\theta)_{\{C\}} \cap R(E)_{\{C\}})\sigma^{-1} \leq E$
$\Rightarrow R(G\theta)_{\{C\}}\sigma^{-1} \leq E$
$\Rightarrow \exists\delta : R(G\theta)_{\{C\}}\sigma^{-1}\delta \subseteq E$
$\Rightarrow (G\theta\sigma'^{-1}\delta')\sigma^{-1}\delta \subseteq E$
$\Rightarrow G\theta' \subseteq E$
◁

Algorithm 1 Sprol

Input: $\underline{E^+}$: positive examples; $\underline{E^-}$: negative examples; $\underline{\alpha}$: the parameter for negative coverage; $\underline{\beta}$: the parameter for positive coverage; $\underline{k}$: the dimension of the population; $\underline{r}$: number of restarts;
Output: the hypotheses h

$C = \text{argmax}_{E_i \in E = E^+ \cup E^-}(|consts(E_i)|)$;
while $E^+ \neq \emptyset$ **do**
  select a seed e from E^+
  /* select k renamings of e */
  Population $\leftarrow ren(k, e, C)$;
  PopPrec $\leftarrow$ Population; $i \leftarrow 0$;
  while $i < r$ **do**
    P $\leftarrow \emptyset$;
    for each element $v \in$ Population **do**
      for each positive example $e^+ \in E^+$ **do**
        /* select t renamings of e^+ */
        $V_{e^+} \leftarrow ren(t, e^+, C)$;
        /* generalization */
        P $\leftarrow$ P $\cup \{u | u = v \cap w_i, w_i \in V_{e^+}\}$;
    Population $\leftarrow$ P;
    /* Consistency check */
    for each negative example $e^- \in E^-$ **do**
      /* select α renamings of e^- */
      $V_{e^-} \leftarrow ren(\alpha, e^-, C)$;
      for each element $v \in$ Population **do**
        if v covers an element of V_{e^-} **then**
          remove v from Population
    /* Completeness check */
    for each element $v \in$ Population **do**
      completeness$_v \leftarrow 0$;
    for each positive example $e^+ \in E^+$ **do**
      /* select β renamings of e^+ */
      $V_{e^+} \leftarrow ren(\beta, e^+, C)$;
      for each element $v \in$ Population **do**
        if $\exists u \in V_{e^+}$ s.t. $u \cap v = v$ **then**
          completeness$_v \leftarrow$ completeness$_v$ + 1;
    $i \leftarrow i + 1$;
    if |Population| $= 0$ **then**
      /* restart with the previous population */
      Population $\leftarrow$ PopPrec;
    else
      leave in Population the best k generalizations only;
      PopPrec $\leftarrow$ Population;
  add the best element $b \in$ Population to h;
  remove from E^+ the positive exs covered by b

To be complete, the procedure must prove the test $G\theta \cap E = G\theta$ for all $P_r^n = \frac{n!}{(n-r)!}$ renamings of $G\theta$ and E, where $n = \max\{|consts(G\theta)|, |consts(E)|\}$ and $r = \min\{|consts(G\theta)|, |consts(E)|\}$ and by taking fixed the renaming for the clause $G\theta$ or E containing less constants. However, we can make the test approximate by randomly choosing a number α of all the possible permutations.

Definition 4 (Subsumption degree). *Let be n the number of all possible renamings of $G\theta$ and E, and $\alpha, \alpha \leq n$, the renamings to test the subsumption between G and E. The subsumption degree between G and E is defined as*

$$sd(G, E) = \begin{cases} \text{argmax}_\alpha \frac{1}{\frac{|R(G\theta)_{\{C\}} \cap R(E)_{\{C\}}|}{|R(G\theta)_{\{C\}}|}} & \text{if } R(G\theta)_{\{C\}} \cap R(E)_{\{C\}} \sim_s G\theta; \\ & \text{otherwise}. \end{cases}$$

In this paper we do not use the subsumption degree to access the validity of generalizations. Each generalization G is considered complete with respect to a positive example E if $R(G\theta)_{\{C\}} \cap R(E)_{\{C\}} \sim_s G\theta$ (*exact completeness*) for a given renaming, and it is considered consistent with respect to a negative example E' if $R(G\theta)_{\{C\}} \cap R(E')_{\{C\}} \sim_s G\theta$ does not hold for all the chosen α renamings (*approximate consistency*). The induction with subsumption degree represents a future work.

To reduce the set of possible permutations we can fix the associations for the variables in the head of the generalization G. In particular if $G : h(V_1, V_2, \ldots, V_d) \leftarrow \ldots$ and $E : h(c_1, c_2, \ldots, c_d) \leftarrow \ldots$ then we can fix in all the generated permutations the associations $\{V_1/c_1, V_2/c_2, \ldots, V_d/c_d\}$, $d \leq r, n$.

Finally, we can further reduce the set of permutations by taking into account the positions of the costants in the literals. Suppose $p(V_1, V_2, \ldots, V_k)$ be a literal of the generalization G. Then, all the constants that may be associated to V_i, $1 \leq i \leq k$, are all those appearing in position i in the literals p/k of the example E.

3.5 Sprol system

Algorithm 1 reports the sketch of the Sprol system, implemented in Yap Prolog 5.1.1, that incorporates ideas of the propositional framework we proposed. Sprol is a population based algorithm where several individual candidate solutions are simultaneously maintained using a constant size population implementing the anytime nature of the algorithm. The population of candidate solutions provides a straightforward means for achieving search diversification and hence for increasing the exploration capabilities of the search process. In our case, the population is made up of candidate generalizations over the training positive examples. In many cases, local minima are quite common in search algorithms and the corresponding candidate solutions are typically not of sufficiently high quality. The strategy we used to escape from local minima is a *restart strategy* that simply reinitializes the search process whenever a local minimum is encountered.

Sprol takes as input the set of positive and negative examples of the training set and some user-defined parameters characterizing its approximate and anytime behaviour. In particular, α and β represent the number of renamings of a negative, respectively positive, example to use for the covering test; k is the size of the population; and r is the number of restarts.

As reported in Algorithm 1, Sprol tries to find a set of clauses that cover all the positive examples and no negative one, by using an iterative population based covering mechanism. It sets the initial population made up of k randomly chosen renamings of a positive example (lines 3-5). Then, the elements of the population are iteratively generalized on the positive examples of the training set (lines 9-15). All the generalizations that cover at least one negative example are taken out (lines 16-22), and the quality of each generalization, based on the number of covered positive examples, is calculated (lines 23-31). Finally, best k generalizations are taken into account for the next iteration (line 37). In case of an empty population a restart is generated with the previous population (line 35).

Renamings of an example are generated according to the procedure reported in Algorithm 2, that randomly chooses k renamings of the example E onto the set of

constants C. This procedure implements the approximate and anytime nature of the method. Indeed, the parameter k represents at the same time both the approximation degree and the time allocated for the algorithm. The more renamings the algorithm select, the more accurate generalizations and subsumptions will be, but the more time to compute them will be needed.

It is important to note that our approach constructs hypotheses that are only approximately consistent. Indeed, in the consistency check it is possible that there exists a matching between an hypothesis and a negative example. The number α of allowed permutations is responsible of the induction cost as well as of the consistency of the produced hypotheses. An obvious consequence is that the more permutations allowed, the more consistent the hypotheses found and, perhaps, the more learning time.

Algorithm 2 $ren(k, E, C)$

Input: $\underline{k}$: the number of renamings; $\underline{E}$: the example; $\underline{C}$: a set of constants;
Output: a set S of renamings of E

1: $S \leftarrow \emptyset$
2: **for** $i = 1$ to k **do**
3: $\quad S \leftarrow S \cup \{R(E)_{\{C\}}\}$

4 Experiments

In order to evaluate the system Sprol, we performed experiments on the classical ILP mutagenesis dataset [17] consisting of structural descriptions of molecules. The Mutagenesis dataset has been collected to identify mutagenic activity in a compound based on its molecular structure and is considered to be a benchmark dataset for multi-relational learning. The Mutagenesis dataset consists of the molecular structure of 230 compounds, of which 138 are labelled as mutagenic and 92 as non-mutagenic. The mutagenicity of the compounds has been determined by the Ames Test. The task is to distinguish mutagenic compounds from non-mutagenic ones based on their molecular structure. The Mutagenesis dataset basically consists of atoms, bonds, atom types, bond types and partial charges on atoms. The dataset also consists of the hydrophobicity of the compound (logP), the energy level of the compound's lowest unoccupied molecular orbital (LUMO), a boolean attribute identifying compounds with 3 or more benzyl rings (I1), and a boolean attribute identifying compounds which are acenthryles (Ia). Ia, I1, logP and LUMO are relevant properties in determining mutagencity.

The size of the population has been set to 50, the parameter α to 50, the parameter β to 50, and making 5 restarts. As measures of performance, we use predictive accuracy and execution time. Results have been compared to those obtained by running, on both the same machine and dataset, the system Progol [18]. A 10-fold cross-validation produced the results reported in Table 2, averaged over the 10-folds, where we can note that there is an evident improvement of the execution time with respect to Progol obtaining a comparable predictive accuracy of the learned theory.

	Progol		SPROL	
	Time	Accuracy	Time	Accuracy
M1	330.76	84.21	56.73	57.89
M2	479.03	78.95	41.15	89.47
M3	535.95	84.21	48.51	73.68
M4	738.54	68.42	63.67	84.21
M5	699.90	89.47	55.56	84.21
M6	497.08	78.95	53.55	78.49
M7	498.22	84.21	71.97	84.21
M8	584.00	78.95	56.29	89.47
M9	511.88	68.42	50.44	83.33
M10	587.18	82.35	65.63	70.59
Mean	546.25	79.81	56.35	79.60

Table 2. Execution time (in seconds) and accuracy of Progol and Sprol on the mutagenesis dataset.

A second experiment, whose result are reported in Table 3, has been made in order to evaluate how the behaviour of the algorithm change by altering parameters k, α and β.

		Time	Accuracy
$\alpha = 50$ $\beta = 50$	$k = 50$	75.49	71.14
	$k = 75$	96.80	75,35
	$k = 100$	117.29	71.67
$\alpha = 50$ $k = 50$	$\beta = 40$	78.84	78.67
	$\beta = 50$	75.49	71.14
	$\beta = 60$	74.39	76.85
	$\beta = 100$	114	78.02
$\beta = 50$ $k = 50$	$\alpha = 40$	75.49	70.19
	$\alpha = 50$	75.49	71.14
	$\alpha = 60$	56.35	79.6

Table 3. Results on parameter settings.

As we can see in Table 3, the first row reports the case in which we fixed α and β and letting k to change. Obviously, taking more elements in the population make grow the execution time. Furtherome, the second and the third row show that changing β does not change the accuracy of the theory. On the contrary α seems to be more important than β in improving the system performances. A further investigation of this behaviour deserve a more accurate experiment on an ad-hoc artificial dataset.

5 Conclusion

Efficient multi-relational data mining algorithms have to tackle the problem of selecting the best search method for exploring the hypotheses space and the problem of reducing

the complexity of the coverage procedure that assessis the validity of the learned theory against the training examples. A way of tackling the complexity of this kind of learning systems is to use a propositional method, that reformulates a multi-relational learning problem into an attribute-value one.

In this paper we proposed a population based algorithm able to efficiently solve multi-relational problems by using an approximate propositional method. The result of an empirical evaluation on the mutagenesis dataset of the proposed technique is very promising and proves the validity of the method.

As a future work, we plan to perform more in-depth experiments, on a purposely defined artificial dataset, in order to evaluate the method dependence from the parameters k, α and β. A solution should be to automatically discover, in an online manner, the correct input parameters of Sprol for a given learning task.

Furthermore, we want to investigate the behaviour of the algorithm in the case of approximate completeness. In particular, we want to use the subsumption degree between clauses in order to induce theories when noisy or uncertain data are available.

Acknowledgements

This work is partially founded by the Italian MIUR FAR project "Laboratorio di Biologia Computazionale per la Biodiversità Molecolare" (Computational Biology Laboratory for Molecular Biodiversity).

References

1. Zilberstein, S., Russell, S.: Approximate reasoning using anytime algorithms. Imprecise and Approximate Computation **318** (1995) 43–62
2. Raedt, L.D.: Attribute value learning versus inductive logic programming: The missing links (extended abstract). In Page, D., ed.: Proceedings of the Eighth International Conference on Inductive Logic Programming. Volume 1446 of LNAI., Springer-Verlag (1998) 1–8
3. Boddy, M., Dean, T.L.: Deliberation scheduling for problem solving in time-constrained environments. Artificial Intelligence **67**(2) (1994) 245–285
4. Lavrac, N., Dzeroski, S., Grobelnik, M.: Learning nonrecursive definitions of relations with linus. In: Proceedings of the European Working Session on Machine Learning, Springer-Verlag (1991) 265–281
5. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York (1994)
6. Krogel, M.A., Rawles, S., Zelezny, F., Flach, P., Lavrac, N., Wrobel, S.: Comparative evaluation of approaches to propositionalization. In Horvath, T., Yamamoto, A., eds.: Proceedings of the 13th International Conference on Inductive Logic Programming. Volume 2835 of LNCS., Springer-Verlag (2003) 194–217
7. Lavrač, N., Železný, F., Flach, P.A.: RSD: Relational subgroup discovery through first-order feature construction. In Matwin, S., Sammut, C., eds.: Proceedings of the 12th International Conference on Inductive Logic Programming. Volume 2583 of LNAI., Springer-Verlag (2002) 149–165
8. Sebag, M., Rouveirol, C.: Induction of maximally general clauses consistent with integrity constraints. In Wrobel, S., ed.: Proceedings of the 4th International Workshop on Inductive Logic Programming. Volume 237 of GMD-Studien., Gesellschaft für Mathematik und Datenverarbeitung MBH (1994) 195–216

9. Zucker, J.D., Ganascia, J.G.: Representation changes for efficient learning in structural domains. In: Proceedings of 13th International Conference on Machine Learning, Morgan Kaufmann (1996) 543–551
10. Alphonse, E., Rouveirol, C.: Lazy propositionalization for relational learning. In Horn, W., ed.: Proc. of the 14th European Conference on Artificial Intelligence, IOS Press (2000) 256–260
11. Dietterich, T., Lathrop, R., Lozano-Perez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artificial Intelligence **89**(1-2) (1997) 31–71
12. Sebag, M., Rouveirol, C.: Tractable induction and classification in first order logic via stochastic matching. In: 15th International Join Conference on Artificial Intelligence, Morgan Kaufmann (1997) 888–893
13. Alphonse, E., Matwin, S.: A dynamic approach to dimensionality reduction in relational learning. In: ISMIS '02: Proceedings of the 13th International Symposium on Foundations of Intelligent Systems. Volume 2366 of LNCS., Springer-Verlag (2002) 255–264
14. Ullman, J.: Principles of Database and Knowledge-Base Systems. Volume I. Computer Science Press (1988)
15. Bratko, I.: Prolog programming for artificial intelligence, 3rd ed. Addison-Wesley Longman Publishing Co. (2001)
16. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. Journal of Logic Programming **19/20** (1994) 629–679
17. Srinivasan, A., Muggleton, S., King, R.: Comparing the use of background knowledge by inductive logic programming systems. In Raedt, L.D., ed.: Proceedings of the 5th International Workshop on Inductive Logic Programming, Springer-Verlag (1995) 199–230
18. Muggleton, S.: Inverse Entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming **13**(3-4) (1995) 245–286