Hybrid Learning of Ontology Classes

1 author:

Jens Lehmann
University of Bonn
290 PUBLICATIONS   **12,317** CITATIONS

Some of the authors of this publication are also working on these related projects:

Project   Structured and Open Question Answering View project

Project   NIF for Open Government Data View project

Hybrid Learning of Ontology Classes

Jens Lehmann

University of Leipzig, Computer Science Department,
Johannisgasse 26, D-04103 Leipzig, Germany,
lehmann@informatik.uni-leipzig.de

Abstract. Description logics have emerged as one of the most successful formalisms for knowledge representation and reasoning. They are now widely used as a basis for ontologies in the Semantic Web. To extend and analyse ontologies, automated methods for knowledge acquisition and mining are being sought for. Despite its importance for knowledge engineers, the learning problem in description logics has not been investigated as deeply as its counterpart for logic programs.

We propose the novel idea of applying evolutionary inspired methods to solve this task. In particular, we show how Genetic Programming can be applied to the learning problem in description logics and combine it with techniques from Inductive Logic Programming. We base our algorithm on thourough theoretical foundations and present a preliminary evaluation.

1 Introduction

Ontologies based on *Semantic Web* technologies are now amongst the most prominent paradigms for knowledge representation. The single most popular ontology language in this context is OWL[1]. However, there is still a lack of available ontologies and tools for creating, extending and analysing ontologies are most demanded. Machine Learning methods for the automated learning of classes from instance data can help to overcome these problems.

While the learning, also referred to as *induction*, of logic programs has been studied extensively in the area of *Inductive Logic Programming* (ILP), the analogous problem for description logics has been investigated to a lesser extend, despite recent efforts [9,11]. This is mainly due to the fact that description logics have only recently become a popular knowledge representation paradigm. The rise of the Semantic Web has increased interest in methods for solving the learning problem in description logics.

Genetic Programming (GP) has been shown to deliver human-competitive machine intelligence in many applications [12]. We show how it can be applied to the learning problem in description logics. Further, we discuss the advantages and drawbacks of this approach and propose a framework for hybrid algorithms combining GP and *refinement operators*.

Refinement operators are central in ILP and we can base our methods on a thorough theoretical analysis of their potential and limitations. We will design

---

[1] http://www.w3.org/2004/OWL

concrete refinement operators and show that they have the desired properties. For the novel algorithm we obtain, a preliminary evaluation is performed, comparing it with standard GP and an other non-evolutionary algorithm for learning in description logics.

The paper is structured as follows. In Section 2 we briefly introduce description logics, Genetic Programming, the learning problem, and refinement operators. Section 3 shows how to apply GP to the learning problem in description logics. The main section is Section 4, in which we show how refinement operators fit in the GP framework. In 5 we report on our prototype implementation. We discuss related work in 6 and draw conclusions in Section 7.

2 Preliminaries

Description Logics Description logics represent knowledge in terms of *objects*, *concepts*, and *roles*. Objects correspond to constants, concepts to unary predicates, and roles to binary predicates in first order logic. In description logic systems information is stored in a *knowledge base*, which is a set of axioms. It is divided in *TBox*, containing *terminology* axioms, and *ABox*, containing *assertional* axioms.

construct	syntax	semantics
concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
existential	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid$
		$\exists b.(a,b) \in r^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}\}$
universal	$\forall r.C$	$(\forall r.C)^{\mathcal{I}} = \{a \mid$
		$\forall b.(a,b) \in r^{\mathcal{I}}$ implies $b \in C^{\mathcal{I}}\}$

Table 1. $\mathcal{ALC}$ syntax and semantics

We briefly introduce the $\mathcal{ALC}$ description logic, which is the target language of our learning algorithm and refer to [3] for further background on description logics. Let N_I denote the set of objects, N_C denote the set of atomic concepts, and N_R denote the set of roles. As usual in logics, interpretations are used to assign a meaning to syntactic constructs. An *interpretation* $\mathcal{I}$ consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$, which assigns to each object $a \in N_I$ an element of $\Delta^{\mathcal{I}}$, to each concept $A \in N_C$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to each role $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Interpretations are extended to elements as shown in Table 1, and to other elements of a knowledge base in a straightforward way. An interpretation, which satisfies an axiom (set of axioms) is called a model of this axiom (set of axioms). An $\mathcal{ALC}$ concept is in *negation normal form* if negation only occurs in front of concept names.

It is the aim of *inference algorithms* to extract implicit knowledge from a given knowledge base. Standard reasoning tasks include *instance checks*, *retrieval* and *subsumption*. We will only explicitly define the latter. Let C, D be concepts and $\mathcal{T}$ a TBox. C *is subsumed by* D, denoted by $C \sqsubseteq D$, iff for any interpretation $\mathcal{I}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. C *is subsumed by* D *with respect to* $\mathcal{T}$ (denoted by $C \sqsubseteq_{\mathcal{T}} D$) iff for any model $\mathcal{I}$ of $\mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. C *is equivalent to* D *(with respect to* $\mathcal{T}$*)*, denoted by $C \equiv D$ ($C \equiv_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and $D \sqsubseteq C$ ($D \sqsubseteq_{\mathcal{T}} C$). C *is strictly subsumed by* D *(with respect to* $\mathcal{T}$*)*, denoted by $C \sqsubset D$ ($C \sqsubset_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and not $C \equiv D$ ($C \equiv_{\mathcal{T}} D$).

Genetic Programming Genetic Programming is one way to automatically solve problems. It is a systematic method to evolve individuals and has been shown to deliver human-competitive machine intelligence in many applications. The distinctive feature of GP within the area of Evolutionary Computing is to represent individuals (not to be confused with individuals in description logics) as variable length programs. In this article, we consider the case that individuals are represented as trees. Inspired by the evolution in the real world, fit individuals are selected from a population by means of different selection methods. New individuals are created from them using genetic operators like crossover and mutation. We do not introduce GP in detail, but instead refer to [12] for more information.

The Concept Learning Problem in Description Logics In this section, we introduce the learning problem in Description Logics. In a very general setting learning means that we have a logical formulation of background knowledge and some observations. We are then looking for ways to extend the background knowledge such that we can explain the observations, i.e. they can be deduced from the modified knowledge.

More formally, we are given background knowledge B, positive examples E^+, negative examples E^- and want to find a hypothesis H such that from H together with B the positive examples follow and the negative examples do not follow. It is not required that the same logical formalism is used for background knowledge, examples, and hypothesis. This means, that although we consider learning $\mathcal{ALC}$ concepts in this article, the background knowledge can be a more expressive description logic.

So let a concept name `Target`, a knowledge base $\mathcal{K}$, and sets E^+ and E^- with elements of the form `Target`(a) $(a \in N_I)$ be given. The learning problem is to find a concept C such that `Target` $\equiv C$ is an acyclic definition and for $\mathcal{K}' = \mathcal{K} \cup \{\text{Target} \equiv C\}$ we have $\mathcal{K}' \models E^+$ and $\mathcal{K}' \not\models E^-$.

For different solutions of the learning problem the simplest ones are to be preferred by the well-known Occam's razor principle [5]. According to this principle, simpler concepts usually have a higher predictive quality. We measure simplicity as the *length* of a concept, which is defined in a straightforward way, namely as the sum of the number of concept, role, quantifier, and connective symbols occurring in the concept.

Refinement Operators Learning can be seen as a search process in the space of concepts. A natural idea is to impose an ordering on this search space and use operators to traverse it. This idea is prominent in Inductive Logic Programming [18], where refinement operators are used to traverse ordered spaces. Downward (upward) refinement operators construct specialisations (generalisations) of hypotheses.

A *quasi-ordering* is a reflexive and transitive relation. In a quasi-ordered space $(S, \preceq)$ a *downward (upward) refinement operator* ρ is a mapping from S to 2^S, such that for any $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$ $(C \preceq C')$. C' is called a *specialisation* (*generalisation*) of C.

Quasi-orderings can be used for searching in the space of concepts. One such quasi-order is subsumption. If a concept C subsumes a concept D $(D \sqsubseteq C)$, then C will cover all examples, which are covered by D. This makes subsumption a suitable order for solving the learning problem.

Definition 1. *A refinement operator in the quasi-ordered space* $(\mathcal{ALC}, \sqsubseteq_{\mathcal{T}})$ *is called an* $\mathcal{ALC}$ *refinement operator.*

We need to introduce some notions for refinement operators. A *refinement chain of an* $\mathcal{ALC}$ *refinement operator* ρ *of length* n from a concept C to a concept D is a finite sequence $C_0, C_1, \ldots, C_n$ of concepts, such that $C = C_0, C_1 \in \rho(C_0), C_2 \in \rho(C_1), \ldots, C_n \in \rho(C_{n-1}), D = C_n$. This refinement chain *goes through* E iff there is an i $(1 \leq i \leq n)$ such that $E = C_i$. We say that D can be reached from C by ρ if there exists a refinement chain from C to D. $\rho^*(C)$ denotes the set of all concepts, which can be reached from C by ρ. $\rho^m(C)$ denotes the set of all concepts, which can be reached from C by a refinement chain of ρ of length m. If we look at refinements of an operator ρ we will often write $C \leadsto_\rho D$ instead of $D \in \rho(C)$. If the used operator is clear from the context it is usually omitted, i.e. we write $C \leadsto D$.

Refinement operators can have certain properties, which can be used to evaluate their usefulness.

Definition 2. *An* $\mathcal{ALC}$ *refinement operator* ρ *is called*

- (locally) finite *iff* $\rho(C)$ *is finite for any concept* C.
- (syntactically) redundant *iff there exists a refinement chain from a concept* C *to a concept* D, *which does not go through some concept* E *and a refinement chain from* C *to a concept weakly equal to* D, *which does go through* E.
- proper *iff for all concepts* C *and* D, $D \in \rho(C)$ *implies* $C \not\equiv D$.
- ideal *iff it is finite, complete (see below), and proper.*
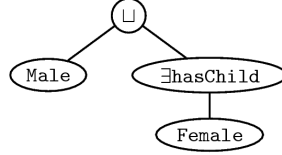
An $\mathcal{ALC}$ *downward refinement operator* ρ *is called*

- complete *iff for all concepts* C, D *with* $C \sqsubset_{\mathcal{T}} D$ *we can reach a concept* E *with* $E \equiv C$ *from* D *by* ρ.
- weakly complete *iff for all concepts* $C \sqsubset_{\mathcal{T}} \top$ *we can reach a concept* E *with* $E \equiv C$ *from* $\top$ *by* ρ.

The corresponding notions for upward refinement operators are defined dually.

3 Concept Learning using Standard GP

To apply GP to the learning problem, we need to be able to represent $\mathcal{ALC}$ concepts as trees. We do this by defining the alphabet $T = N_C \cup \{\top, \bot\}$ and $F = \{\sqcup, \sqcap, \neg\} \cup \{\forall r \mid r \in N_R\} \cup \{\exists r \mid r \in N_R\}$, where T is the set of terminal symbols and F the set of function symbols.

Example 1. The $\mathcal{ALC}$ concept `Male` $\sqcup$ $\exists$`hasChild.Female` can be represented as the following tree:

We say that an alphabet has the *closure* property if any function symbol can handle as an argument any data type and value returned by an alphabet symbol. Using the presented encoding the closure property is satisfied, because the way trees are build corresponds exactly to the inductive definition of $\mathcal{ALC}$ concepts. This ensures that tree operations like crossover and mutation can be performed safely, i.e. the obtained trees also represent $\mathcal{ALC}$ concepts.

Fitness Measurement To be able to apply GP to the learning problem we need to define a fitness measure. To do this, we introduce some notions.

Definition 3 (covered examples). *Let* `Target` *be the target concept, $\mathcal{K}$ a knowledge base, and C an arbitrary $\mathcal{ALC}$ concept. The set of positive examples covered by C, denoted by $pos_{\mathcal{K}}(C)$, is defined as:*

$$pos_{\mathcal{K}}(C) = \{\textit{Target}(a) \mid a \in N_I, K \cup \{\textit{Target} \equiv C\} \models \textit{Target}(a)\} \cap E^+$$

Analogously, the set of negative examples covered by C, denoted by $neg_{\mathcal{K}}(C)$, is defined as:

$$neg_{\mathcal{K}}(C) = \{\textit{Target}(a) \mid a \in N_I, K \cup \{\textit{Target} \equiv C\} \models \textit{Target}(a)\} \cap E^-$$

Of course, the fitness measurement should give credit to covered positive examples and penalize covered negative examples. In addition to these classification criteria, it is also useful to bias the GP algorithm towards shorter solutions. A possible fitness functions is:

$$f_{\mathcal{K}}(C) = -\frac{|E^+ \setminus pos_{\mathcal{K}}(C)| + |neg_{\mathcal{K}}(C)|}{|E^+| + |E^-|} - a \cdot |C| \quad (0 < a < 1)$$

The parameter a is the decline in classification accuracy one is willing to accept for a concept, which is shorter by one length unit. Being able to represent solutions and measuring their fitness is already sufficient to apply Genetic Programming to a problem. We discuss some advantages and problems of doing this.

Advantages GP is a very flexible learning method. It is not only able to learn in $\mathcal{ALC}$, but can also handle other description languages (languages with role constructors can be handled using the framework of Strongly Typed GP). GP has been shown to deliver good results in practice and is especially suited in situations, where approximate solutions are acceptable [13]. An additional advantage is that GP algorithms are parallelizable and can make use of computational resources, i.e. if more resources (time and memory) exist its parameters can be changed to increase the probability of finding good solutions. This may seem obvious, but in fact this does not hold for many (deterministic) solution methods. GP also allows for a variety of extensions and is able to handle noise naturally (the parameter a in the introduced fitness function is one way to handle noise).

Problems of the Standard Approach Despite the described advantages of GP, there are some notable drawbacks. One problem is that the crossover operator is too destructive. For GP to work well, it should be the case that high fitness individuals are likely to produce high fitness offspring. (This is the reason why selection methods are used instead of random selection.) For crossover on $\mathcal{ALC}$ concepts, small changes in a concept can drastically change its semantics, so it is not very likely that the offspring of high fitness individuals also has a high fitness. Similar problems arise when using GP in ILP and indeed a lot of systems use non-standard operators [7].

Another problem of the standard approach is that we do not use all knowledge we have. An essential insight in Machine Learning [15] is that the approaches, which use most knowledge about the learning problem they want to solve, usually perform best. The standard GP algorithm does not make use of subsumption as quasi-order on concepts. Thus, a natural idea is to enhance the standard GP algorithm by operators, which exploit the subsumption order.

4 Refinement Operators in Genetic Programming

4.1 Transforming Refinement Operators to Genetic Refinement Operators

As argued before, it is useful to modify the standard GP approach to make learning more efficient. The idea we propose is to integrate refinement operators in GP. This aims to resolve the two problems we have outlined above: Well-designed refinement operators are usually less destructive, because applying such an operator to a concept means that only a small change to the concept is performed – syntactically and semantically. Moreover, refinement operators can make use of the subsumption order and, thus, use more available knowledge than the standard GP algorithm. We show how refinement operators and GP can be combined in general and then present a concrete operator.

Some steps need to be done in order to be able to use refinement operators as genetic operators. The first problem is that a refinement operator is a mapping from one concept to an arbitrary number of concepts. Naturally, the idea is to

select one of the possible refinements. In order to be able to do this efficiently, we assume that the refinement operators we are looking at are finite.

The second problem when applying refinement operators to GP is that a concrete refinement operator only performs either specialisation or generalisation, but not both. However, in GP we are likely to find too strong as well as too weak concepts, so there is a need for upward and downward refinement. A simple approach is to use two genetic operators: an adapted upward and an adapted downward refinement operator.

Another way to solve the problem is to use one genetic operator, which stochastically chooses whether downward or upward refinement is used. This allows to adjust the probabilities of upward or downward refinement being selected to the classification of the concept we are looking at. For instance consider an overly general concept, i.e. it covers all positive examples, but does also cover some negative examples. In this case we always want to specialize, so the probability for using downward refinement should be 1. In the opposite case for an overly specific concept, i.e. none of the negatives is covered, but some positives, the probability of downward refinement should be 0. How do we assign probabilities to concepts, which are neither overly specific nor overly general? Our approach is as follows:

1. The probability of downward refinement, denoted by $p_\downarrow$, should depend on the percentage of covered negative examples. Using α as variable factor we get:
$$p_\downarrow(\mathcal{K}, C) = \alpha \cdot \frac{|neg_\mathcal{K}(C)|}{|E^-|}$$
   In particular for $|neg_\mathcal{K}(C)| = 0$ (consistent concept) we get $p_\downarrow(\mathcal{K}, C) = 0$.
2. The probability of upward refinement, denoted by $p_\uparrow$, should depend on the percentage of covered positive examples. We use the same factor as in the first case:
$$p_\uparrow(\mathcal{K}, C) = \alpha \cdot \left(1 - \frac{|pos_\mathcal{K}(C)|}{|E^+|}\right)$$
   In particular for $|pos_\mathcal{K}(C)| = |E^+|$ (complete concept) we get $p_\uparrow(\mathcal{K}, C) = 0$.
3. For any concept $p_\downarrow(\mathcal{K}, C) + p_\uparrow(\mathcal{K}, C) = 1$.

From this, we can derive the following formulae for the probabilities of upward and downward refinement:

$$p_\downarrow(\mathcal{K}, C) = \frac{\frac{|neg_\mathcal{K}(C)|}{|E^-|}}{1 + \frac{|neg_\mathcal{K}(C)|}{|E^-|} - \frac{|pos_\mathcal{K}(C)|}{|E^+|}} \quad p_\uparrow(\mathcal{K}, C) = \frac{1 - \frac{|pos_\mathcal{K}(C)|}{|E^+|}}{1 + \frac{|neg_\mathcal{K}(C)|}{|E^-|} - \frac{|pos_\mathcal{K}(C)|}{|E^+|}}$$

Note that the return value of the formula is undefined, due to division by zero, for complete and consistent concepts. However, in this case C is a learning problem solution and we can stop the algorithm – or continue it to find smaller solutions by just randomly selecting whether upward or downward refinement is used.

This way we have given a possible solution to both problems: transforming the refinement operator to a mapping from a concept to exactly one concept

and managing specialisation and generalisation. Overall, for a given finite upward refinement operator $\phi_\uparrow$ and a finite downward refinement operator $\phi_\downarrow$ we can construct a genetic operator ϕ, which is defined as follows (rand selects an element of a given set uniformly at random):

$$\phi_{\mathcal{K}}(C) = \begin{cases} \text{rand}(\phi_\downarrow(C)) & \text{with probability } \dfrac{\frac{|neg_{\mathcal{K}}(C)|}{|E^-|}}{1+\frac{|neg_{\mathcal{K}}(C)|}{|E^-|}-\frac{|pos_{\mathcal{K}}(C)|}{|E^+|}} \\ \text{rand}(\phi_\uparrow(C)) & \text{with probability } \dfrac{1-\frac{|pos_{\mathcal{K}}(C)|}{|E^+|}}{1+\frac{|neg_{\mathcal{K}}(C)|}{|E^-|}-\frac{|pos_{\mathcal{K}}(C)|}{|E^+|}} \end{cases} \quad (1)$$

In the sequel, we will call genetic operators, which are created from upward and downward refinement operators this way, *genetic refinement operators*.

4.2 A Genetic Refinement Operator

To design a suitable refinement operator for learning $\mathcal{ALC}$ concepts, we first look at theoretical limitations. The following theorem [14] is a full analysis of the properties of $\mathcal{ALC}$ refinement operators:

Theorem 1. *Considering the properties completeness, weak completeness, properness, finiteness, and non-redundancy the following are maximal sets of properties (in the sense that no other of the mentioned properties can be added) of $\mathcal{ALC}$ refinement operators:*

1. *{weakly complete, complete, finite}*
2. *{weakly complete, complete, proper}*
3. *{weakly complete, non-redundant, finite}*
4. *{weakly complete, non-redundant, proper}*
5. *{non-redundant, finite, proper}*

We prefer complete operators, because this guarantees that, by applying the operator, we always have the possibility to find a solution of the learning problem. As argued before, we also need a finite operator. This means that a complete and finite operator is the best we can hope for. We will define such an operator in the sequel.

For $A \in N_C$ and background knowledge $K = (\mathcal{T}, \mathcal{A})$, we define $\text{nb}_\downarrow(A) = \{A' \mid A' \in N_C, \text{ there is no } A'' \in N_C \text{ with } A' \sqsubset_{\mathcal{T}} A'' \sqsubset_{\mathcal{T}} A\}$. $\text{nb}_\uparrow(A)$ is defined analogously. Furthermore, we define the operator $\phi_\downarrow$ as shown in Figure 1. It works on concepts in negation normal form, so an input concept has to be converted if necessary.

Proposition 1. $\phi_\downarrow$ *is an $\mathcal{ALC}$ downward refinement operator.*

Proof. We show that $D \in \phi_\downarrow(C)$ implies $D \sqsubseteq_{\mathcal{T}} C$ by structural induction over $\mathcal{ALC}$ concepts in negation normal form. We can ignore refinements of the form $C \rightsquigarrow C \sqcap \top$, because obviously $C \sqsubseteq_{\mathcal{T}} C \sqcap \top$ ($C \equiv_{\mathcal{T}} C \sqcap \top$). We also ignore refinements of the form $C \rightsquigarrow \bot$, for which the claim is also true. All other cases are shown below.

$$\phi_\downarrow(C) = \begin{cases} \emptyset & \text{if } C = \bot \\ \{\forall r.\top \mid r \in N_R\} \cup \{\exists r.\top \mid r \in N_R\} \cup \{\top \sqcup \top\} & \text{if } C = \top \\ \quad \cup \{A \mid \text{nb}_\uparrow(A) = \emptyset\} \cup \{\neg A \mid \text{nb}_\downarrow(A) = \emptyset\} & \\ \{A' \mid A' \in \text{nb}_\downarrow(A)\} \cup \{\bot \mid \text{nb}_\downarrow(A) = \emptyset\} \cup \{A \sqcap \top\} & \text{if } C = A \ (A \in N_C) \\ \{\neg A' \mid A' \in \text{nb}_\uparrow(A)\} \cup \{\bot \mid \text{nb}_\uparrow(A) = \emptyset\} \cup \{\neg A \sqcap \top\} & \text{if } C = \neg A \ (A \in N_C) \\ \{\exists r.E \mid E \in \phi_\downarrow(D)\} \cup \{\exists r.D \sqcap \top\} \cup \{\bot \mid D = \bot\} & \text{if } C = \exists r.D \\ \{\forall r.E \mid E \in \phi_\downarrow(D)\} \cup \{\forall r.D \sqcap \top\} \cup \{\bot \mid D = \bot\} & \text{if } C = \forall r.D \\ \{C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \cdots \sqcap C_n & \text{if } C = C_1 \sqcap \cdots \sqcap C_n \\ \quad \mid D \in \phi_\downarrow(C_i), 1 \le i \le n\} & (n \ge 2) \\ \quad \cup \{C_1 \sqcap \cdots \sqcap C_n \sqcap \top\} & \\ \{C_1 \sqcup \cdots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \cdots \sqcup C_n & \text{if } C = C_1 \sqcup \cdots \sqcup C_n \\ \quad \mid D \in \phi_\downarrow(C_i), 1 \le i \le n\} & (n \ge 2) \\ \quad \cup \{C_1 \sqcup \cdots \sqcup C_{i-1} \sqcup C_{i+1} \sqcup \cdots \sqcup C_n \mid 1 \le i \le n\} & \\ \quad \cup \{(C_1 \sqcup \cdots \sqcup C_n) \sqcap \top\} & \end{cases}$$

Fig. 1. definition of $\phi_\downarrow$

- $C = \bot$: $D \in \phi_\downarrow(C)$ is impossible, because $\phi_\downarrow(\bot) = \emptyset$.
- $C = \top$: $D \sqsubseteq_\mathcal{T} C$ is trivially true for each concept D (and hence also for all refinements).
- $C = A \ (A \in N_C)$: $D \in \phi_\downarrow(C)$ implies that D is also an atomic concept or the bottom concept and $D \sqsubset C$.
- $C = \neg A$: $D \in \phi_\downarrow(C)$ implies that D is of the form $\neg A'$ with $A \sqsubseteq_\mathcal{T} A'$. $A \sqsubseteq_\mathcal{T} A'$ implies $\neg A' \sqsubseteq_\mathcal{T} \neg A$ by the semantics of negation.
- $C = \exists r.C'$: $D \in \phi_\downarrow(C)$ implies that D is of the form $\exists r.D'$. We have $D' \sqsubseteq_\mathcal{T} C'$ by induction. For existential restrictions $\exists r.E \sqsubseteq_\mathcal{T} \exists r.E'$ if $E \sqsubset_\mathcal{T} E'$ holds in general [4]. Thus we also have $\exists r.D' \sqsubseteq \exists r.C'$.
- $C = \forall r.C'$: This case is analogous to the previous one. For universal restrictions $\forall r.E \sqsubseteq_\mathcal{T} \forall r.E'$ if $E \sqsubset_\mathcal{T} E'$ holds in general [4].
- $C = C_1 \sqcap \cdots \sqcap C_n$: In this case one element of the conjunction is refined, so $D \sqsubseteq_\mathcal{T} C$ follows by induction.
- $C = C_1 \sqcup \cdots \sqcup C_n$: One possible refinement is to apply $\phi_\downarrow$ to one element of the disjunction, so $D \sqsubseteq_\mathcal{T} C$ follows by induction. Another possible refinement is to drop an element of the disjunction, when $D \sqsubseteq_\mathcal{T} C$ obviously also holds.

Proposition 2. $\phi_\downarrow$ is complete.

Proof. We will first show weak completeness of $\phi_\downarrow$. We do this by structural induction over $\mathcal{ALC}$ concepts in negation normal form, i.e. we show that every concept in negation normal form can be reached by $\phi_\downarrow$ from $\top$.

- Induction Base:
  - $\top$: $\top$ can trivially be reached from $\top$.
  - $\bot$: $\top \rightsquigarrow A_1 \rightsquigarrow \ldots \rightsquigarrow A_n \rightsquigarrow \bot$ (descending the subsumption hierarchy)
  - $A \in N_C$: $\top \rightsquigarrow A_1 \rightsquigarrow \ldots \rightsquigarrow A_n \rightsquigarrow A$ (descending the subsumption hierarchy until A is reached)
  - $\neg A (A \in N_C)$: $\top \rightsquigarrow \neg A_1 \rightsquigarrow \ldots \rightsquigarrow \neg A_n \rightsquigarrow \neg A$ (ascending the subsumption hierarchy of atomic concepts within the scope of a negation symbol)

- Induction Step:
    - $\exists r.C$: $\top \rightsquigarrow \exists r.\top \rightsquigarrow^* \exists r.C$ (last step by induction)
    - $\forall r.C$: $\top \rightsquigarrow \forall r.\top \rightsquigarrow^* \forall r.C$ (last step by induction)
    - $C_1 \sqcap \cdots \sqcap C_n$: $\top \rightsquigarrow^* C_1$ (by induction) $\rightsquigarrow C_1 \sqcap \top \rightsquigarrow^* C_1 \sqcap C_2 \rightsquigarrow^* C_1 \sqcap \cdots \sqcap C_n$
    - $C_1 \sqcup \cdots \sqcup C_n$: $\top \rightsquigarrow \top \sqcup \top \rightsquigarrow^* C_1 \sqcup \top$ (by induction) $\rightsquigarrow C_1 \sqcup \top \sqcup \top \rightsquigarrow^* C_1 \sqcup C_2 \sqcup \top \rightsquigarrow^* C_1 \sqcup \cdots \sqcup C_n$

We have shown that $\phi_\downarrow$ is weakly complete. If we have two $\mathcal{ALC}$ concepts C and D in negation normal form with $C \sqsubseteq_\mathcal{T} D$, then for a concept $E = D \sqcap C$ we have $E \equiv_\mathcal{T} C$. E can be reached by the following refinement chain from D:

$$D \rightsquigarrow D \sqcap \top \rightsquigarrow^* D \sqcap C \text{ (by weak completeness of } \phi_\downarrow)$$

Thus, we have shown that we can reach a concept equivalent to C, which proves the completeness of $\phi_\downarrow$.

Proposition 3. $\phi_\downarrow$ *is finite.*

Proof. Some rules in the definition of $\phi_\downarrow$ apply $\phi_\downarrow$ recursively, e.g. specialising an element of a conjunction. Since such applications are only performed on inner structures of an input concept, only finitely many recursions are necessary to compute all refinements. This means that it is sufficient to show that every single application of $\phi_\downarrow$ produces finitely many refinements under the assumption that each recursive application of $\phi_\downarrow$ on an inner structure represents a finite set. Since N_R and N_C are finite, this can be verified easily by analysing all cases in Figure 1.

We have shown that $\phi_\downarrow$ is complete and finite, which makes it suitable to be used in a genetic refinement operator. We defined a dual upward refinement operator $\phi_\uparrow$ and showed its completeness and finiteness. The definition of the operator and the proofs are omitted, because they are analogous to what we have shown for $\phi_\downarrow$. From $\phi_\downarrow$ and $\phi_\uparrow$ we can construct a genetic refinement operator as described in Equation 1. This new operator is ready to be used within the GP framework and combines classical induction with evolutionary approaches.

What are the differences between classical refinement operator based approaches and our evolutionary approach? Usually, in a classical algorithm a refinement operator spans a search tree and a search heuristic guides the direction of search. The heuristic corresponds to the fitness function in a GP and usually both bias the search towards small concepts with high classification accuracy.

The search space in a classical algorithm is traversed in a well-structured and often deterministic manner. However, such an algorithm has to maintain (parts of) a search tree, which is usually continuously growing. In a GP, the population has, in most cases, a constant size. This means that a GP can run for a long time without consuming more space (assuming that the individuals themselves are not constantly growing, which is not the case for our genetic refinement operator). In this sense, a GP can be seen as a less structured search with individuals moving

stochastically in the search space. Another difference is that classical algorithms often traverse the search space only in one direction (bottom-up or top-down approach), whereas genetic refinement operators use both directions and can start from random points in the search space. In general, it is not clear whether a classical or hybrid approach is to be preferred and the choice – as usual in Machine Learning – depends on the specific problem at hand.

5 Preliminary Evaluation

To perform a preliminary evaluation, we have chosen the FORTE [20] family data set. We transformed it into an OWL ontology about family relationships and defined a new learning task. In our case, the ontology contains two disjoint concepts `Male` and `Female`, the roles `parent`, `sibling` (symmetric), and `married` (symmetric and functional). The family tree is described by 337 assertional axioms. As learning target, we have chosen the concept of an uncle. A possible definition of this concept is:

$$\texttt{Male} \sqcap (\exists\,\texttt{sibling}.\exists\,\texttt{parent}.\top \sqcup \exists\,\texttt{married}.\exists\,\texttt{sibling}.\exists\,\texttt{parent}.\top)$$

86 examples, 23 positive and 63 negative, are provided. This learning task can be considered challenging, since the smallest possible solution is long (length 13) and there are no restrictions on the search space. For our experiments, we have chosen to let the GP algorithm run a fixed number of 50 generations. We used a generational algorithm and initialised it using the ramped-half-and-half method with maximum depth 6 for initialisation. The fitness measure in Section 3 with a non-optimised value of $a = 0.0025$ was used, i.e. a length unit is worth a accuracy decline of 0.25% . As selection method, we have chosen rank selection defined in such a way that the highest ranked individual has a ten times higher probability of being selected than the lowest ranked individual.
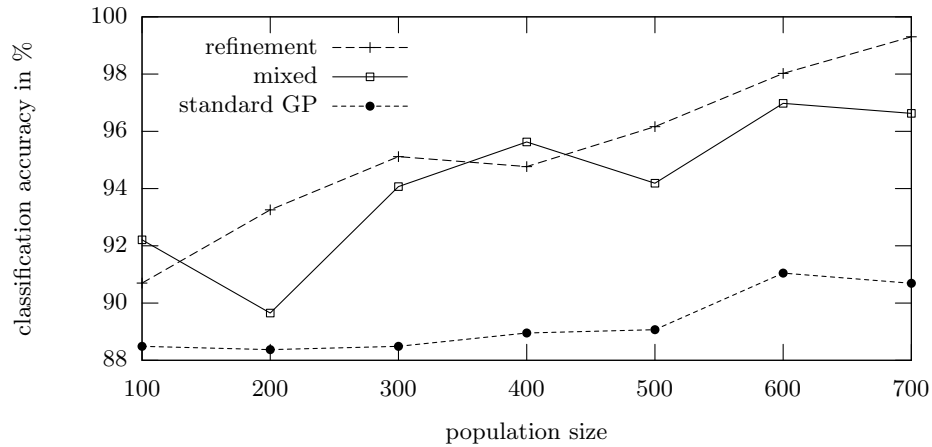

Fig. 2. classification accuracy on family data set

Since our main contribution is the provision of a new operator, we have tested three sets of operator probabilities. All sets have a 2% probability for mutation.

The standard GP set has an 80% probability of crossover (the remaining 18% are used for reproduction). The mixed set uses 40% crossover and 40% genetic refinement operator and the refinement set uses only 5% crossover and 85% genetic refinement operator. We have varied the population size from 100 to 700 and averaged all results over 10 runs. Figure 2 depicts the results we obtained with respect to classification accuracy of the examples (defined as in the first part of Equation 3). Under the assumptions of a t test, the difference in accuracy for standard GP compared to one of the two others is statistically significant with a confidence interval of 95% for population sizes higher than 200.

Apart from the classification accuracy, we also measured the length of the concepts, which were returned as solutions by the algorithm. The results are shown in Figure 3. All algorithms were always able to find at least a concept of length 3 with a classification accuracy of 88%. Since the number of $\mathcal{ALC}$ concepts grows exponentially with their length, it is much harder for a learning algorithm to find promising long concepts. In most cases, standard GP failed to do so, whereas the other algorithms had high success rates for high population sizes.
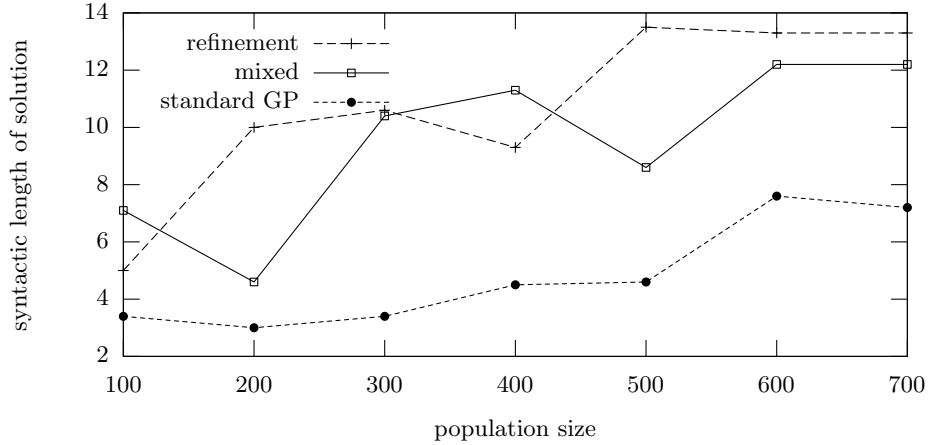

Fig. 3. length of learned concepts on family data set

For the experiments we used the reasoner Pellet[2] (version 1.4RC1), which was connected to the learning program using the DIG 1.1 interface[3]. Most of the runtime of the algorithm (98%) was spent for reasoner requests. Since the genetic refinement operator is not proper and performs only small changes on concepts, we built up a caching mechanism for it. Before saving a concept in the cache, we normalized it. First, we defined a linear order over $\mathcal{ALC}$ concepts and ordered elements in conjunctions and disjunctions according to this order. Additionally, we converted the concept to negation normal form and applied equivalence preserving rewriting rules e.g. $C \sqcap \top \to C$. This techniques allowed us to use the cache for approximately 65% of the computed refinements of the genetic refinement operator. Due to more cache hits, the performance of the

---

[2] http://pellet.owldl.com
[3] http://dl.kr.org/dig/

genetic refinement operator is even better than for crossover and mutation. The overall runtime varied from approximately 100 seconds on average for a population size of 100 to 950 seconds on average for a population size of 700 on a 2,2 GHz dual core processor[4] with 2 GB RAM.

The YinYang system [11] has a runtime of 200 seconds for this example, a classification accuracy of 73.5%, and a concept length of 12.2 averaged over 10 runs[5]. We could not use other systems for comparison. The system in [6] is not available anymore and the approach in [4] was not fully implemented.

6   Related Work

To the best of our knowledge, there has been no attempt to use evolutionary approaches for learning concepts in description logics. Hence, there is no closely related work we are aware of. Approaches for concept learning in description logics are described in [6,4,9,11]. Although evolutionary methods have not been considered for learning in description logics before, they have been used for inducing logic programs. A recent article [7] provides a good overview.

Evolutionary ILP systems usually use variants of Genetic Algorithms or Genetic Programming. The goal is to learn a set of clauses for a target predicate. EVIL_1 [19] is a system based on Progol [16], where an individual represents a set of clauses (called the *Pittsburgh approach*) and crossover operators are used. REGAL [17] is a system, which consists of a network of genetic nodes to achieve high parallelism. Each individual encodes a partial solution (called the *Michigan approach*). It uses classic mutation and several crossover operators. GNET is a descendant of REGAL. It also uses a network of genetic nodes, but takes a co-evolutionary approach [1], i.e. two algorithms are used to converge to a solution. DOGMA [10] is a system, which uses a combination of the Pittsburgh and Michigan approach on different levels of abstraction. All these systems use a simple bit string representation. This is possible by requiring a fixed template, which the solution must fit in. We did not consider this approach when learning in description logics due to its restricted flexibility.

The following systems use a high level representation of individuals. SIA01 [2] is a bottom-up approach, which starts with a positive example as seed and grows a population until it reaches a bound (so the population size is not fixed as in the standard approach). ECL [8] is a system using only mutation style operators for finding a solution. In contrast GLPS [23] uses only crossover style operators and a tree (more exactly forest) representation of individuals. In [22] a binary representation of clauses is introduced, which is shown to be processable by genetic operators in a meaningful way. [21] extends this framework by a fast fitness evaluation algorithm.

The systems based on Genetic Programming, i.e. SIA01, ECL, and GLPS are closest to our approach. Similar to our research, they also concluded that

---

[4] Our current GP implementation does not efficiently use the second CPU.

[5] 3 out of 86 examples could not be used, because YinYang could not calculate most specific concepts for them, which are needed as input for their algorithm.

standard GP is not sufficient to solve their learning problem. As a consequence, they invented new operators. As far as we know, they did not try to connect refinement operators and GP explicitly as we did. We cannot directly compare the operators, which are used in ILP systems, with the genetic operator we have developed, since the target language (logic programs) is different.

7   Conclusions, Further Work

In the article, we have presented a hybrid approach for learning concepts in DLs, which combines GP and refinement operators. We first presented how to solve the problem using standard GP, outlined difficulties and showed how they can be overcome using refinement operators. To the best of our knowledge, this is the first time a framework for transforming refinement operators to genetic operators has been proposed and the first time that evolutionary techniques have been applied to the concept learning problem in description logics. Based on a full property analysis [14], we developed a concrete genetic refinement operator and provided a preliminary evaluation.

In the future we plan to extend our evaluation, propose benchmark data sets for the learning problem, and analyse the interaction between genetic refinement operators and traditional operators.

References

1. Cosimo Anglano, Attilio Giordana, Giuseppe Lo Bello, and Lorenza Saitta. An experimental evaluation of coevolutive concept learning. In *Proc. 15th International Conf. on Machine Learning*, pages 19–27. Morgan Kaufmann, 1998.
2. S. Augier, G. Venturini, and Y. Kodratoff. Learning first order logic rules with a genetic algorithm. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *The First International Conference on Knowledge Discovery and Data Mining*, pages 21–26, Montreal, Canada, 20-21 August 1995. AAAI Press.
3. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
4. Liviu Badea and Shan-Hwei Nienhuys-Cheng. A refinement operator for description logics. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 40–59. Springer-Verlag, 2000.
5. Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*, pages 201–204. Morgan Kaufmann, 1990.
6. William W. Cohen and Haym Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR94)*, pages 121–133. Morgan Kaufmann, 1994.
7. Federico Divina. Evolutionary concept learning in first order logic: An overview. *AI Commun.*, 19(1):13–33, 2006.

8. Federico Divina and Elena Marchiori. Evolutionary concept learning. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 343–350, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

9. Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-intensive induction of terminologies from metadata. In *ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Proceedings*, pages 441–455. Springer, 2004.

10. Jukka Hekanaho. Background knowledge in GA-based concept learning. In *Proc. 13th International Conference on Machine Learning*, pages 234–242. Morgan Kaufmann, 1996.

11. Luigi Iannone and Ignazio Palmisano. An algorithm based on counterfactuals for concept learning in the semantic web. In Moonis Ali and Floriana Esposito, editors, *Innovations in Applied Artificial Intelligence*, pages 370–379, Bari, Italy, June 2005. Proceedings of the 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems.

12. John R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

13. John R. Koza and Riccardo Poli. A genetic programming tutorial. In Edmond Burke, editor, *Introductory Tutorials in Optimization, Search and Decision Support*, 2003.

14. Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. Technical report, University of Leipzig, 2007.

15. Thomas Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.

16. Stephen Muggleton. Inverse entailment and progol. *New Generation Computing*, 13(3&4):245–286, 1995.

17. Filippo Neri and Lorenza Saitta. Analysis of genetic algorithms evolution under pure selection. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 32–39. Morgan Kaufmann, 1995.

18. Shan-Hwei Nienhuys-Cheng and Ronald de Wolf, editors. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997.

19. Philip G. K. Reiser and Patricia J. Riddle. Evolution of logic programs: Part-of-speech tagging. In *1999 Congress on Evolutionary Computation*, pages 1338–1345, Piscataway, NJ, 1999. IEEE Service Center.

20. B. L. Richards and R. J. Mooney. Refinement of first-order Horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

21. A. Tamaddoni-Nezhad and S. Muggleton. A genetic algorithms approach to ILP. In S. Matwin and C. Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 285–300. Springer-Verlag, 2003.

22. Alireza Tamaddoni-Nezhad and Stephen Muggleton. Searching the subsumption lattice by a genetic algorithm. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 243–252. Springer-Verlag, 2000.

23. Man Leung Wong and Kwong Sak Leung. Inducing logic programs with genetic algorithms: The genetic logic programming system. *IEEE Expert*, 10 5:68–76, October 1995.