Using Genetic Algorithms for Learning Clauses in First-Order Logic

Alireza Tamaddoni-Nezhad
Department of Computer Science
University of York, York, YO10 5DD, UK
alireza@cs.york.ac.uk

Stephen Muggleton
Department of Computer Science
University of York, York, YO10 5DD, UK
stephen@cs.york.ac.uk

Abstract

A framework for combining first-order concept learning with Genetic Algorithms is introduced. This framework includes: 1) a novel binary representation for clauses 2) task-specific genetic operators 3) a fast evaluation mechanism. The proposed binary representation encodes the refinement space of clauses in a natural and compact way. It is shown that essential operations on clauses such as unification and anti-unification can be done by simple bitwise operations (e.g. and/or) on the binary encoding of clauses. These properties are used for designing task-specific genetic operators. It is also shown that by using these properties individuals can be evaluated at genotype level without mapping them into corresponding clauses. This replaces the complex task of evaluating clauses, which usually needs repeated theorem proving, by simple bitwise operations. An implementation of the proposed framework is used to combine Inverse Entailment of the learning system CProgol with a genetic search.

1 INTRODUCTION

In concept learning problems, there is a trade-off between the expressive power of the representation and the complexity of hypotheses search space. In the case of first-order concept learning this search space grows combinatorially and the search is usually intractable. On the other hand, many real-world applications require a representation language which is at least as expressive as first-order logic. For example, Inductive Logic Programming (ILP) (Muggleton, 1991; Muggleton and Raedt, 1994) has been a fast growing research area in the last decade and has shown many successes in machine learning and data-mining applications, especially in some challenging domains such as bioinformatics (Muggleton et al., 1992; Srinivasan et al., 1997; Muggleton, 1999). Current first-order learning systems mostly employ deterministic search methods to examine the refinement space of clauses and use different kind of syntactic biases and heuristics (e.g. greedy methods) to cope with the complexity of the search which otherwise is intractable. Using these biases usually limits the exploration power of the search and may lead to local optima. Hence, more powerful search methods are required for inducing complex concepts and for dealing with massive data. Genetic Algorithms (GAs) have great potential for this purpose. GAs are multi-point search methods (and less sensitive to local optima) which can search through a large space and manipulate symbolic as well as numerical data. Moreover, because of their robustness and adaptive characteristics, GAs are suitable methods for optimization and learning in many real world applications (Goldberg, 1989). In terms of implementation, GAs are highly parallel and can be easily implemented in parallel and/or distributed models. However, GAs are syntactically restricted and cannot represent a priori knowledge that already exists about the domain. On the other hand, first-order concept learning methods, such as ILP, are well known paradigms that benefit from the expressive power inherited from logic and logic programming. Hence, it is likely that a combination of such learning paradigms and GAs can overcome the limitation of each individual method and can be used to cope with some complexities of real-world applications.

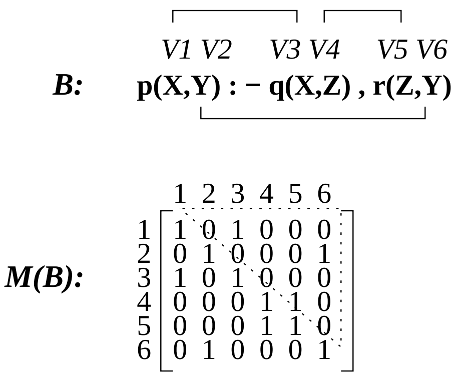
Even though GAs have been used widely for optimization and learning in many domains, a few genetic-based systems in first-order domain already exist. Some of these systems (Giordana and Sale, 1992; Giordana and Neri, 1996; Hekanaho, 1998; Anglano et al.,

1998) follow conventional genetic algorithms and represent problem solutions by fixed length bit-strings. Other systems (Varšek, 1993; Leung and Wong, 1995; Wong and Leung, 1997; Kennedy and Giraud-Carrier, 1999; Reiser and Riddle, 1998) use a hierarchical representation and evolve a population of logic programs in a Genetic Programming (GP) (Koza, 1991) manner. These genetic-based systems confirm that genetic algorithms and evolutionary computing can be interesting alternatives for learning first-order concepts from examples. However, these systems mostly use genetic search as the only learning mechanism in the system and hence they cannot benefit from the existing first-order learning techniques for example for utilizing background knowledge in the learning process.

This paper aims to present a framework for combining first-order concept learning with GAs by introducing a novel binary encoding for clauses, relevant genetic operators and a fast evaluation mechanism. In this framework, unlike other genetic-based systems, representation and operators can be interpreted in well known first-order logic terms such as subsumption, unification and anti-unification (Plotkin, 1969; Nienhuys-Cheng and de Wolf, 1997). This representation and its interpretations and properties are introduced in the next section. Section 3 shows how some properties of the binary representation can be used to design task-specific genetic operators. In this section we show how essential operations on clauses can be done by simple bitwise operations (e.g. and/or) on binary strings. As another property of the proposed representation we show that evaluating a clause, which is a complex task, can also be done by simple and fast bitwise operations. This evaluation mechanism is introduced in section 4. Section 5 describes an implementation of the proposed framework for combining Inverse Entailment in CProgol (Muggleton, 1995) with a genetic algorithm. Evaluations and related works are discussed in section 6. Finally, section 7 summarizes the results and concludes this paper.

2 REPRESENTATION AND ENCODING

Every application of GAs requires formulating problem solutions in such a way that they can be processed by genetic operators. It has been suggested that the binary representation is a suitable coding for representing problem solutions in GAs (Holland, 1975; Goldberg, 1989). The lack of a proper (binary) representation, and consequently difficulties for definition and implementation of genetic operators, has been the main problem for applying GAs in first-order domain.



$$
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 1 & 0 & 0 & 0 & 1 \\
3 & 1 & 0 & 1 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 1 & 1 & 0 \\
5 & 0 & 0 & 0 & 1 & 1 & 0 \\
6 & 0 & 1 & 0 & 0 & 0 & 1
\end{array}
$$

Figure 1: Binding matrix for clause *p(X,Y):-q(X,Z),r(Z,Y)*.

In this section we introduce a novel binary representation for first-order clauses. We show that this representation encodes the refinement space of clauses in a natural and compact way. Even though all definition and theorems in this paper hold for the general form of firs-order clauses, for simplicity and consistency reasons, in all examples we use Horn clauses [1] which is the standard representation in logic programming. Consider a clause with n variable occurrences. The relationships between these n variable occurrences can be represented by a graph having n vertices in which there exists an edge between vertices v_i and v_j if ith and jth variable occurrences in the clause represent the same variable. For example variable bindings in clause *p(X,Y):-q(X,Z),r(Z,Y)* represent an undirected graph and this clause can be represented by a binary matrix as shown in Figure 1. In this matrix entry m_{ij} is 1 if ith and jth variable occurrences in the clause represent the same variable and m_{ij} is 0 otherwise. This representation has interesting properties which can be exploited by a genetic algorithm for searching the refinement space of a clause. Before we formally show these properties, we need to show the mappings between clauses and binary matrices.

Definition 1 (Binding Set) *Let B and C both be clauses. C is in binding set $\mathcal{B}(B)$ if there exists a variable substitution [2] θ such that $C\theta = B$.*

In Definition 1, the variables in B induce a set of equivalence classes over the variables in any clause

[1] In Horn clauses all clauses contain at most one positive literal. For example $\{p(X,Y) \vee \sim q(X,Z) \vee \sim r(Z,Y)\}$ is a Horn clause and can be represented by *p(X,Y):-q(X,Z),r(Z,Y)*.

[2] Substitution $\theta = \{v_i/u_j\}$ is a variable substitution if all v_i and u_j are variables.

$C \in \mathcal{B}(B)$. Thus we could write the equivalence class of u in variable substitution θ as $[v]_u$, the set of all variables in C such that v/u is in θ. We define a binary matrix which represents whether variables v_i and v_j are in the same equivalence class or not.

Definition 2 (Binding Matrix) *Suppose B and C are both clauses and there exists a variable substitution θ such that $C\theta = B$. Let C have n variable occurrences representing variables $\langle v_1, v_2, \ldots, v_n \rangle$. The binding matrix of C is an $n \times n$ matrix M in which m_{ij} is 1 if there exist variables v_i, v_j and u such that v_i/u and v_j/u are in θ and m_{ij} is 0 otherwise. We write $M(v_i, v_j) = 1$ if $m_{ij} = 1$ and $M(v_i, v_j) = 0$ if $m_{ij} = 0$.*

Definition 3 (Normalized Binding Matrix) *Let M be an $n \times n$ binary matrix. M is in the set of normalized binding matrices $\mathcal{M}_n$ if M is symmetric and for each $1 \leq i \leq n$, $1 \leq j \leq n$ and $1 \leq k \leq n$, $m_{ij} = 1$ if $m_{ik} = 1$ and $m_{kj} = 1$.*

Definition 4 (Mapping Function M(C)) *The mapping function $M : \mathcal{B}(B) \rightarrow \mathcal{M}_n$ is defined as follows. Given clause $C \in \mathcal{B}(B)$ with n variable occurrences representing variables $\langle v_1, v_2, \ldots, v_n \rangle$, $M(C)$ is an $n \times n$ binary matrix in which m_{ij} is 1 if variables v_i and v_j are identical and m_{ij} is 0 otherwise.*

Definition 5 (Mapping Function C(M)) *The mapping function $C : \mathcal{M}_n \rightarrow \mathcal{B}(B)$ is defined as follows. Given a normalized $n \times n$ binding matrix M, $C(M)$ is a clause in $\mathcal{B}(B)$ with n variable occurrences $\langle v_1, v_2, \ldots, v_n \rangle$, in which variables v_i and v_j are identical if m_{ij} is 1.*

Definition 6 (Matrix Subset) *Let P and Q be in $\mathcal{M}_n$. It is said that $P \subseteq Q$ if for each entry $p_{ij} \in P$ and $q_{ij} \in Q$, p_{ij} is 1 if q_{ij} is 1. $P = Q$ if $P \subseteq Q$ and $Q \subseteq P$. $P \subset Q$ if $P \subseteq Q$ and $P \neq Q$.*

Definition 7 (Clause Subsumption) *Clause C subsumes clause D, $C \succeq D$ if there exists a (variable) substitution θ such that $C\theta \subseteq D$ (i.e. every literal in $C\theta$ is also in D). C properly subsumes D, $C \succ D$ if $C \succeq D$ and $D \not\succeq C$.*

Because of the subsumption order between clauses (which is a quasi-order) the search space (or refinement space) can be modeled as a subsumption lattice (Nienhuys-Cheng and de Wolf, 1997). The following theorem represents the relationship between binary matrices and the subsumption order of clauses.

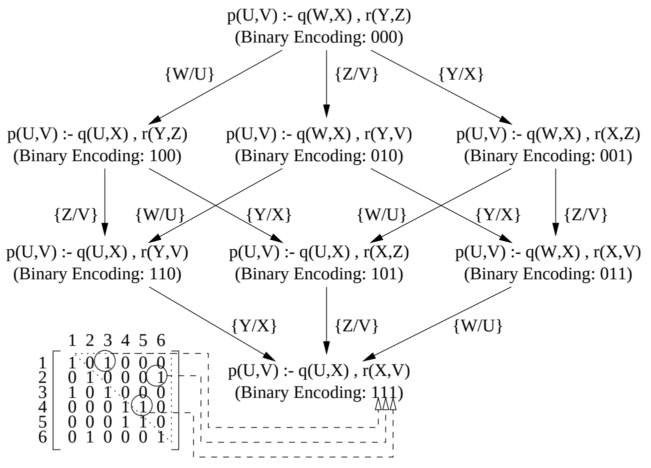


Figure 2: A subsumption lattice bounded below by clause *p(X,Y):-q(X,Z),r(Z,Y)* and binary encoding for each clause.

Theorem 1 *For each clause B and matrices M_1 and M_2 in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$ and $C(M_2) \in \mathcal{B}(B)$, $C(M_1) \succ C(M_2)$ if $M_1 \subset M_2$.*

Proof. Suppose $M_1 \subset M_2$. Therefore there exist variables v_i and v_j, $i < j$ such that $M_1(v_i, v_j) = 0$ and $M_2(v_i, v_j) = 1$. Then according to Definition 2, $C(M_1)\{v_i/v_j\} = C(M_2)$. Hence, $C(M_1) \succ C(M_2)$. $\square$

A binding matrix is a symmetric matrix in which diagonal entries are 1. In practice, we only maintain entries in top (or down) triangle of the matrix. Furthermore, in our implementation (see section 5), we are interested in a subsumption lattice bounded below by a particular clause. Hence, each member of $\mathcal{B}(B)$ can be encoded by a binary string in which each bit corresponds to a 1 entry of matrix $M(B)$.

Example 1 *Figure 2 shows a subsumption lattice bounded below by the clause p(X,Y):-q(X,Z),r(Z,Y). Each clause in this search space can be encoded by 3 bits.*

3 GENETIC OPERATORS AND STOCHASTIC REFINEMENT

Genetic operators introduce new individuals into population by randomly changing or combining the genotype of best-fit individuals during an evolutionary process. In conventional genetic algorithms these operators are domain-independent and usually without any assumption about the problem on hand. However, more efficient genetic operators can be designed by using simple facts about the domain. For example it has been shown that introducing generalization and specialization crossover operators, which are used together with standard crossover and mutation operators, can be useful in concept learning problems (Giordana and Sale, 1992; Janikow, 1993). In this section we show that the proposed binary representation has great potential for designing task-specific genetic operators. In particular, we show how mgi(most general instance) and lgg(least general generalization) which are also known as unification and anti-unification operations on clauses (Plotkin, 1969; Nienhuys-Cheng and de Wolf, 1997) can be achieved by simple bitwise operations on the binary encoding of clauses. In the following, first we introduce *mgi* and *lgg* operations for clauses.

Definition 8 (mgi and lgg) *Clauses E and F are respectively a common instance and a common generalization of clauses C and D if and only if $C, D \succeq E$ and $F \succeq C, D$. $mgi(C,D)$ and $lgg(C,D)$ are the most general instance and the least general generalization for clauses C and D if and only if for every common instance E and common generalization F it is the case that $mgi(C,D) \succeq E$ and $F \succeq lgg(C,D)$.*

Example 2 *In Figure 2 clause p(U,V):-q(U,X),r(X,Z) is the mgi of clauses p(U,V):-q(U,X),r(Y,Z) and p(U,V):-q(W,X),r(X,Z) and clause p(U,V):-q(W,X),r(Y,V) is the lgg of clauses p(U,V):-q(U,X),r(Y,V) and p(U,V):-q(W,X),r(X,V).*

Definition 9 (Matrix AND) *Let M_1 and M_2 be in $\mathcal{M}_n$. $M = (M_1 \wedge M_2)$ is an $n \times n$ matrix and for each $a_{ij} \in M$, $b_{ij} \in M_1$ and $c_{ij} \in M_2$, $a_{ij} = 1$ if $b_{ij} = 1$ and $c_{ij} = 1$ and $a_{ij} = 0$ otherwise.*

Similar to AND operator, OR operator $(M_1 \vee M_2)$ is constructed by bitwise OR-ing of M_1 and M_2 entries.

Definition 10 (Matrix OR) *Let M_1 and M_2 be in $\mathcal{M}_n$. $M = (M_1 \vee M_2)$ is an $n \times n$ matrix and for each $a_{ij} \in M$, $b_{ij} \in M_1$ and $c_{ij} \in M_2$, $a_{ij} = 1$ if $b_{ij} = 1$ or $c_{ij} = 1$ and $a_{ij} = 0$ otherwise.*

Theorem 2 *For each clause B and matrices M_1, M_2 and M in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$, $C(M_2) \in \mathcal{B}(B)$ and $C(M) \in \mathcal{B}(B)$, $C(M) = lgg(C(M_1), C(M_2))$ if $M = (M_1 \wedge M_2)$.*

Proof. Suppose $M = M_1 \wedge M_2$. Therefore $M \subset M_1$ and $M \subset M_2$ and according to Theorem 1 $C(M) \succ C(M_1)$ and $C(M) \succ C(M_2)$. Therefore $C(M)$ is a common generalization of $C(M_1)$ and $C(M_2)$. We show that $C(M)$ is the least general generalization of $C(M_1)$ and $C(M_2)$. For each binding matrix M' in $\mathcal{M}_n$ it must be the case that if $C(M') \succ C(M_1)$ and $C(M') \succ C(M_2)$ then $C(M') \succ C(M)$. Suppose $C(M') \not\succ C(M)$ then according to Theorem 1 there exist u and v such that $M'(u,v) = 1$ and $M(u,v) = 0$. If $M'(u,v) = 1$ then $M_1(u,v) = 1$ and $M_2(u,v) = 1$ and this contradicts $M(u,v) = 0$ and completes the proof. □

By a similar proof it can be shown that the result of or-operator is equivalent to *mgi.*

Theorem 3 *For each clause B and matrices M_1, M_2 and M in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$, $C(M_2) \in \mathcal{B}(B)$ and $C(M) \in \mathcal{B}(B)$, $C(M) = mgi(C(M_1), C(M_2))$ if $M = M_1 \vee M_2$.*

Proof. Symmetric with proof of Theorem 2. □

Example 3 *In Figure 2, lgg and mgi of any two clauses can be obtained by AND-ing and OR-ing of their binary strings.*

According to these theorems unification and anti-unification can be done by simple bitwise operations on the binary encoding of clauses. These properties can be used for designing task-specific genetic operators such as generalization and specialization crossover operators. Generalization and specialization are known as the main operations in concept learning methods (Winston, 1970; Mitchell, 1982; Mitchell, 1997). In particular, *lgg* and *mgi* are essential in first-order learning. For example, the ILP system Golem (Muggleton and Feng, 1990) which was successfully applied to a wide range real-world applications (Bratko et al., 1991; Feng, 1992; Muggleton et al., 1992) only uses a *lgg* operator which operates on determinacy restricted clauses [3]. In addition to the generalization and specialization crossovers mentioned earlier, we can also introduce task-specific mutation operators. In the standard mutation operator we use a fixed probability (mutation-rate) for changing 0 and 1 bits [4]. As shown in the previous section, the difference between bits in binding matrices determines the subsumption order between clauses. Hence, the subsumption distance between clauses increases monotonically with the Ham-

[3]It has been shown that the determinacy restriction is inappropriate in some applications (Muggleton, 1994). There is not such a restriction for the *lgg* operator introduced in this paper.

[4]A random mutation could result in a matrix which is not consistent with Definition 3. Even though this inconsistency doesn't affect the genetic search in practice, it could be avoided by a normalization closure using Definition 3.

ming distance between corresponding matrices. We can use this property to set different mutation rates for 0 and 1 bits based on a desirable degree of generalization and specialization. This leads to a directed mutation operator.

In first-order concept learning, upward and downward refinement operators are used for generalization and specialization of clauses (Nienhuys-Cheng and de Wolf, 1997). In our case, task-specific genetic operators can be interpreted as *stochastic refinement operators* in the context of first-order concept learning.

4 FAST EVALUATION AT GENOTYPE LEVEL

The usual way for evaluating a hypothesis in first-order concept learning systems is to repeatedly call a theorem prover (e.g. Prolog interpreter) on training examples to find out positive and negative coverage of the hypothesis. This step is known to be a complex and time-consuming task in first-order concept learning. In the case of genetic-based systems this situation is even worse, because we need to evaluate a population of hypotheses in each generation. This problem is another important difficulty when applying GAs in first-order concept learning.

In this section we introduce a method which replaces the complex task of evaluating clauses by bitwise operations on binary strings. This idea is similar to data-compilation method used by the attribute-based learning system GIL (Janikow, 1993). This system retains binary coverage vectors for all possible features (attributes and values) which can appear in a rule. This introduces a database which can be used for computing the coverage set of each rule by bitwise operations on the coverage vectors of participating features. However, in our case there is no need to maintain such a database. We show that by maintaining the coverage sets for a small number of clauses and by doing bitwise operations we can compute the coverage for other binary strings without mapping them into the corresponding clauses. This property is based on the implicit subsumption order which exists in the binary representation. In the following, first we define the cover-vector approach for representing coverage of a clause on training examples.

Definition 11 (Cover Sets and Cover Vectors) *Let C be a clause and $E^+ = \{e_1^+, e_2^+, \ldots, e_k^+\}$ and $E^- = \{e_1^-, e_2^-, \ldots, e_l^-\}$ be the set of positive and negative training examples respectively. e_i^+ is in the positive cover set $\mathcal{P}(C)$ if $C \succeq e_i^+$. Similarly, e_j^- is in the negative cover set $\mathcal{N}(C)$ if $C \succeq e_j^-$. The positive cover vector $\mathcal{PV}(C)$ is a k-bit binary string in which bit i is 1 if $e_i^+ \in \mathcal{P}(C)$ and 0 otherwise. Similarly, the negative cover vector $\mathcal{NV}(C)$ is a l-bit binary string in which bit j is 1 if $e_j^- \in \mathcal{N}(C)$ and 0 otherwise.*

Theorem 4 *For each clause C_1 and C_2, $\mathcal{P}(mgi(C_1, C_2)) = \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$.*

Proof. Let $e \in \mathcal{P}(mgi(C_1, C_2))$, then according to Definition 11, $mgi(C_1, C_2) \succeq e$. But according to the definition of mgi, $C_1 \succeq mgi(C_1, C_2)$ and $C_2 \succeq mgi(C_1, C_2)$ and therefore $C_1 \succeq e$ and $C_2 \succeq e$. Hence, $e \in \mathcal{P}(C_1)$ and $e \in \mathcal{P}(C_2)$ and therefore $e \in \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$. Hence, $\mathcal{P}(mgi(C_1, C_2)) \subset \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$. Now, let $e \in \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$, then according to Definition 11, $C_1 \succeq e$ and $C_2 \succeq e$. But according to the definition of mgi, $mgi(C_1, C_2) \succeq e$. Hence, $e \in \mathcal{P}(mgi(C_1, C_2))$ and therefore $\mathcal{P}(C_1) \cap \mathcal{P}(C_2) \subset \mathcal{P}(mgi(C_1, C_2))$ and this completes the proof. $\Box$

Theorem 5 *For each M, M_1 and M_2 in $\mathcal{M}_n$, $\mathcal{PV}(C(M)) = \mathcal{PV}(C(M_1)) \wedge \mathcal{PV}(C(M_2))$ if $M = M_1 \vee M_2$.*

Proof. Suppose $M = M_1 \vee M_2$. Therefore according to Theorem 3, $C(M) = mgi(C(M_1), C(M_2))$. Then according to Theorem 4 ,$\mathcal{P}(C(M)) = \mathcal{P}(C(M_1)) \cap \mathcal{P}(C(M_2))$. But according to Definition 11, $\mathcal{PV}(C(M)) = \mathcal{PV}(C(M_1)) \wedge \mathcal{PV}(C(M_2))$. $\Box$

This theorem, which also holds for negative coverage vectors, can be easily extended for n clauses. According to these theorems, positive (or negative) coverage of clauses can be computed by bitwise operations. Hence, the evaluation of each individual is done at genotype level without mapping it into the corresponding phenotype (clause).

Example 4 *In Fig. 2, $\mathcal{PV}(C(111)) = \mathcal{PV}(C(110)) \wedge \mathcal{PV}(C(101))$.*

5 IMPLEMENTATION

In our first attempt, we employed the proposed representation to combine Inverse Entailment in CProgol4.4 with a genetic algorithm. CProgol is an Inductive Logic Programming (ILP) system which develops first-order hypotheses from examples and background knowledge. CProgol uses Inverse Entailment (Muggleton, 1995) to construct the most specific clause (or the bottom-clause) for each example and then searches for the best clause H which subsumes this bottom-clause ($\Box \succeq H \succeq \bot$). This introduces a subsumption lattice bounded below by the bottom-clause ($\perp$). The standard CProgol starts from the empty clause ($\square$) and uses an A^*-like algorithm for searching this bounded subsumption lattice. The details for CProgol's A^*-like search, refinement operator and algorithm for building the bottom-clause can be found in (Muggleton, 1995).

As shown in section 2, $\mathcal{B}(\perp)$ represents a subsumption lattice bounded below by the bottom-clause. We used a genetic algorithm together with the binary encoding for clauses (as described in section 2) to evolve a randomly generated population of binary strings in which each individual corresponds to a member of $\mathcal{B}(\perp)$. Because of simple representation and straightforward operators any standard genetic algorithm can be used for this purpose. We used a *Simple Genetic Algorithm(SGA)* (Goldberg, 1989) and modified it to suit the representation introduced in this paper. This genetic search evolves a population of hypotheses which all subsume the bottom-clause and uses an evaluation function which is similar to one used in the A^*-like search of CProgol [5] but normalized between 0 and 1. The predicates which violate the mode declaration language are considered as inactive predicates which can be filtered from the induced hypothesis.

In our first experiment, we applied the genetic search to learn Michlaski's east-bound trains (Michalski, 1980). Figure 3 compares the performance of the genetic search with a random search in which each population is generated randomly as in the initial generation. In all experiments (10/10) a correct solution was discovered by the genetic search before generation 20 (standard deviations for the average fitness mean over 10 runs are shown as error bars). The following parameter setting was used for SGA: $popsize = 30$, $p_m = 0.0333$ and $p_c = 0.6$. Figure 3 also compares the convergence of the standard SGA with a SGA which uses together with standard one-point crossover, the task-specific operators lgg introduced in section 3. The following parameter setting was used for $SGA + lgg$: $popsize = 30$, $p_m = 0.0333$, $p_c = 1 - \alpha * f$ and $p_{lgg} = \alpha * f$ where f is the mean value of the fitness of the parental strings ($f = \frac{f(s_1)+f(s_2)}{2}$) and $\alpha = 0.8$ [6].

Preliminary results show that the A^*-like search exhibits better performance in learning hypotheses with small and medium complexities. However, the performance of the genetic search is less dependent on

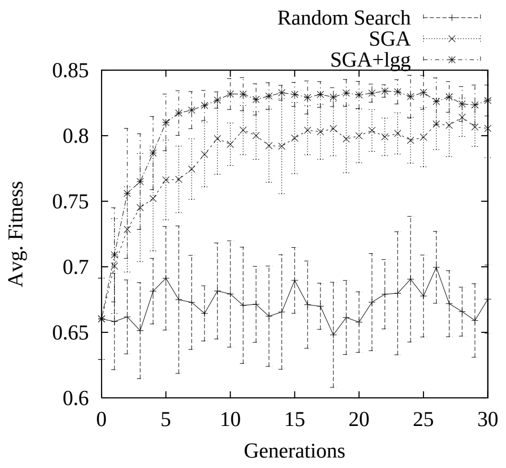


Figure 3: Convergence of the genetic search in the trains problem.

the complexity of hypotheses, whereas A^*-like search shows a great dependency on this factor. Moreover, genetic search can find the correct solution for some special cases which the solution is beyond the exploration power of the A^*-like search due to its incompleteness (Muggleton, 1995; Badea and Stanciu, 1999).

6 DISCUSSION AND RELATED WORKS

The actual completeness and complexity exhibited by the standard A^*-like search of CProgol depends upon the order of literals in the bottom clause and upon the complexity of the hypothesis. In contrast, the genetic search is less dependent on the complexity of the hypothesis and is not affected by the order of literals in the bottom clause. Therefore it is reasonable that genetic algorithm is able to find some solutions which are beyond the exploration power of the A^*-like search.

As mentioned earlier, one main difficulty in order to apply GAs in first-order domain concerns formulating first-order hypotheses into bit-strings. GA-SMART (Giordana and Sale, 1992) was the first relation learning system which tackled this problem by restricting concept description language and introducing a language template. A template in GA-SMART is a fixed length CNF formula which must be defined by the user. Mapping a formula into bit-string is done by setting the corresponding bits to represent the occurrences of predicates in the formula. The main problem of this method is that the number of conjuncts in the template grows combinatorially with the number of predicates. REGAL (Giordana and Neri, 1996),

[5]The criteria used in the evaluation function of the A^*-like search of CProgol include: number of positive and negative examples covered by the clause, length of the clause and number of further literals to complete the clause. More details can be found in (Muggleton, 1995).

[6]This parameter setting is similar to one used in (Giordana and Sale, 1992).

DOGMA (Hekanaho, 1998) and G-NET (Anglano et al., 1998) mainly follow the same idea of GA-SMART and employ a user-defined template for mapping first-order rules into bit strings. However, instead of using a standard representation, a template in these systems is a conjunction of internally disjunctive predicates. This leads to some difficulties for example for representing continuous attributes. Other systems including GILP (Varšek, 1993), GLPS (Leung and Wong, 1995), LOGENPRO (Wong and Leung, 1997), STEPS (Kennedy and Giraud-Carrier, 1999) and EVIL (Reiser and Riddle, 1998) use hierarchical representations rather than using fixed length bit-strings. These systems evolve a population of logic programs in a Genetic Programming (GP) (Koza, 1991) manner. Even though some of the above mentioned systems use background knowledge for generating initial population or seeding the population, most of these systems cannot benefit from intentional background knowledge as it is used in usual first-order learning systems. On the other hand, in our proposed framework, encoding of hypotheses is based on a most specific or bottom-clause which is constructed according to the background knowledge and training examples. This bottom-clause can be automatically constructed using logic-based methods such as Inverse Entailment. Moreover, as shown in section 2 and section 3, the proposed encoding and operators can be interpreted in well known first-order logic terms.

7 CONCLUSIONS AND FURTHER WORK

In this paper we have introduced a framework for combining first-order concept learning with GAs. Efficient binary representation for encoding clauses and its properties, relevant task-specific operators and the fast evaluation mechanism are the major novelty of the proposed framework.

A preliminary implementation of this framework is used to combine Inverse Entailment of the ILP system CProgol with a genetic search. Even though this implementation justifies the properness of the proposed framework, it could be improved in many ways. A natural improvement might be using more sophisticated genetic algorithms rather than using a simple genetic algorithm. For example the greedy cover set algorithm of CProgol, which repeatedly generalizes examples, could be replaced by a distributed genetic algorithm. The task-specific genetic operators can be used to guide the genetic search towards the interesting areas of the search space by specialization and/or generalization as it is done in usual concept learning systems. The fast evaluation mechanism can be used to compensate for the natural computation cost of a genetic algorithm and could lead to a high performance genetic search. In the current approach, the occurrence of atoms in a clause is not considered in the binary encoding of the clause and inactive atoms (e.g. unconnected predicates) are filtered from the induced hypotheses. This could lead to an incomplete search or inexact evaluation. Alternatively, the presence or absence of atoms in each clause can be encoded as a part of the binary representation of the clause. Finally, more experiments are required to evaluate the proposed framework in complex domains and noisy domains.

Undoubtedly, first-order concept learning and GAs are on opposite sides in the classification of learning processes (Kodratoff and Michalski, 1990). While GAs are known as empirical or BK-poor, first-order concept learning could be considered as BK-intensive method in this classification. We conclude that the framework proposed in this paper can be considered as a *bridge* between these two different paradigms to utilize the distinguishable benefits of each method in a hybrid system.

References

Anglano, C., Giordana, A., Bello, G. L., and Saitta, L. (1998). An experimental evaluation of coevolutive concept learning. In Shavlik, J., editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 19–27. Morgan Kaufmann.

Badea, L. and Stanciu, M. (1999). Refinement operators can be (weakly) perfect. In Džeroski, S. and Flach, P., editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 21–32. Springer-Verlag.

Bratko, I., Muggleton, S., and Varsek, A. (1991). Learning qualitative models of dynamic systems. In *Proceedings of the Eighth International Machine Learning Workshop*, San Mateo, Ca. Morgan-Kaufmann.

Feng, C. (1992). Inducing temporal fault dignostic rules from a qualitative model. In Muggleton, S., editor, *Inductive Logic Programming*. Academic Press, London.

Giordana, A. and Neri, F. (1996). Search-intensive concept induction. *Evolutionary Computation Journal*, 3(4):375–416.

Giordana, A. and Sale, C. (1992). Learning structured concepts using genetic algorithms. In Sleeman, D. and Edwards, P., editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 169–178. Morgan Kaufmann.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA.

Hekanaho, J. (1998). Dogma: A ga-based relational learner. In Page, D., editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, pages 205–214. Springer-Verlag.

Holland, J. (1975). *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228.

Kennedy, C. J. and Giraud-Carrier, C. (1999). An evolutionary approach to concept learning with structured data. In *Proceedings of the fourth International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 1–6. Springer Verlag.

Kodratoff, Y. and Michalski, R. (1990). Research in machine learning: Recent progress, classification of methods and future directions. In Kodratoff, Y. and Michalski, R., editors, *Machine learning: an artificial intelligence approach*, volume 3, pages 3–30. Morgan Kaufman, San Mateo, CA.

Koza, J. R. (1991). *Genetic Programming.* MIT Press, Cambridge, MA.

Leung, K. S. and Wong, M. L. (1995). Genetic logic programming and applications. *IEEE Expert*, 10(5):68–76.

Michalski, R. (1980). Pattern recognition as rule-guided inductive inference. In *Proceedings of IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 349–361.

Mitchell, T. (1982). Generalisation as search. *Artificial Intelligence*, 18:203–226.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill, New York.

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295–318.

Muggleton, S. (1994). Inductive logic programming: derivations, successes and shortcomings. *SIGART Bulletin*, 5(1):5–11.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, 13:245–286.

Muggleton, S. (1999). Scientific knowledge discovery using Inductive Logic Programming. *Communications of the ACM*, 42(11):42–46.

Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo. Ohmsha.

Muggleton, S., King, R., and Sternberg, M. (1992). Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657.

Muggleton, S. and Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679.

Nienhuys-Cheng, S.-H. and de Wolf, R. (1997). *Foundations of Inductive Logic Programming.* Springer-Verlag, Berlin. LNAI 1228.

Plotkin, G. (1969). A note on inductive generalisation. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, Edinburgh.

Reiser, P. G. K. and Riddle, P. J. (1998). Evolving logic programs to classify chess-endgame positions. In Newton, C., editor, *Second Asia-Pacific Conference on Simulated Evolution and Learning*, Canberra, Australia.

Srinivasan, A., , Muggleton, R. K. S., and Sternberg, M. (1997). The predictive toxicology evaluation challenge. In *Proceedings of the Fifteenth International Joint Conference Artificial Intelligence (IJCAI-97)*, pages 1–6. Morgan-Kaufmann.

Varšek, A. (1993). *Inductive Logic Programming with Genetic Algorithms.* PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia.

Winston, P. H. (1970). *Learning Structural Descriptions from Examples.* Phd thesis, MIT, Cambridge, Massachusetts.

Wong, M. L. and Leung, K. S. (1997). Evolutionary program induction directed by logic grammars. *Evolutionary Computation*, 5(2):143–180.