Searching the Subsumption Lattice by a Genetic Algorithm

Alireza Tamaddoni-Nezhad and Stephen H. Muggleton

Department of Computer Science
University of York, York, YO1 5DD, UK
{alireza,stephen}@cs.york.ac.uk

Abstract. A framework for combining Genetic Algorithms with ILP methods is introduced and a novel binary representation and relevant genetic operators are discussed. It is shown that the proposed representation encodes a subsumption lattice in a complete and compact way. It is also shown that the proposed genetic operators are meaningful and can be interpreted in ILP terms such as lgg(least general generalization) and mgi(most general instance). These operators can be used to explore a subsumption lattice efficiently by doing binary operations (e.g. and/or). An implementation of the proposed framework is used to combine Inverse Entailment of CProgol with a genetic search.

1 Introduction

Using complete and efficient methods for searching the refinement space of hypothesis is a challenging issue in current ILP systems. Different kinds of greedy methods as well as heuristics (e.g. information gain) have been successfully employed to cope with complexity of search for inducing first-order concepts from examples. However, more powerful heuristics are required for inducing complex concepts and for searching very large search spaces. 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 of 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 [6, 4]. In terms of implementation, GAs are highly parallel and can be easily implemented in parallel and/or distributed environments [4]. However, GAs are syntactically restricted and cannot represent a priori knowledge that already exists about the domain. On the other hand, ILP is a well known paradigm that benefits from the expressive power inherited from logic and logic programming [12]. Hence, it is likely that a combination of ILP 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 [3, 2, 5] follow conventional genetic algorithms and represent problem solutions by fixed length bit-strings. Other systems [16, 10, 7] use a hierarchical representation and evolve a population of logic programs in a Genetic Programming (GP) [9] manner. These genetic-based systems confirm that genetic algorithms and evolutionary computing can be interesting alternatives for learning first-order concepts from examples. However, in most of these systems genetic algorithm is the main learning mechanism and therefore they cannot benefit from background knowledge during the learning process.

This paper aims to present a framework for combining GAs with ILP methods by introducing a novel binary encoding and relevant genetic operators. In this framework, unlike other genetic-based systems, representation and operators can be interpreted in well known ILP terms such as θ-subsumption, lgg (least general generalization) and mgi(most general instance) [14, 15]. This representation and its ILP interpretation are introduced in the next section. Genetic operators and their relationship with ILP concepts are examined in section 3. Section 4 describes an implementation of the proposed framework for combining Inverse Entailment in CProgol [13] with a genetic algorithm. Evaluations and related works are discussed in section 5. Finally, section 6 summarizes the results and concludes this paper.

2 Representation and Encoding

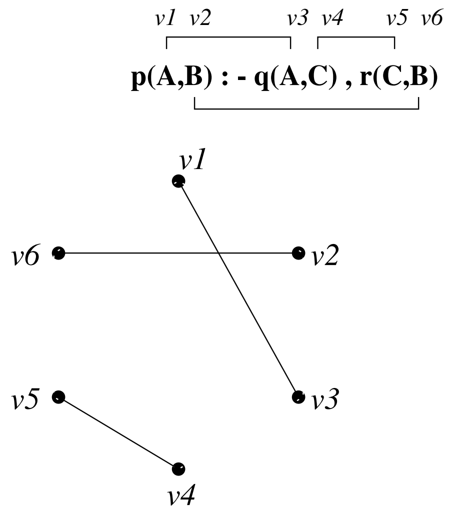
Every application of GAs requires formulating problem solutions in such a way that they can be processed by genetic operators. According to *the principle of minimal alphabet* in GAs [4], a binary representation could be the best coding for representing problem solutions in GAs. The lack of a proper binary representation (and consequently difficulties for definition and implementation of genetic operators) has been the main problem for applying GAs in first-order domain.

In this section we present a binary representation which encodes the refinement space of a clause. Consider clause C with n variable occurrences in head and body. 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 binding in clause *p(A,B):-q(A,C),r(C,B)* can be represented by an undirected graph as shown in Figure 1, this clause also can be represented by the binary matrix shown in the figure. 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. We show that this simple representation has interesting properties for searching the subsumption lattice bounded below by clause C. First, 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* [1] *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 $C \in \mathcal{B}(B)$. Thus we could write the equivalence class of

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



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

Fig. 1. Binding Graph and Binding Matrix for clause *p(A,B):-q(A,C),r(C,B).*

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 in head and body 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. *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 $1 \le i \le n$, $1 \le j \le n$ and $1 \le k \le n$, $m_{ij} = 1$ if $m_{ik} = 1$ and $m_{kj} = 1$.*

Definition 4. *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 in head and body 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. *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. *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 C subsumes clause D, $C \succeq D$ if there exists a 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 $D \not\succeq C$.*

In Definition 7, θ can be every substitution, however, in this paper we assume that θ is a variable substitution. The following theorems represent the relationship between binary matrices and the subsumption of clauses.

Lemma 1. *For each M_1 and M_2 in $\mathcal{M}_n$, if $C(M_1) \succ C(M_2)$ and there does not exist clause C' such that $C(M_1) \succ C' \succ C(M_2)$ then there exists unique $\langle v_i, v_j \rangle$, $i < j$ such that $M_1(v_i, v_j) = 0$ and $M_2(v_i, v_j) = 1$ and for each u' and v', $M_1(u', v') = M_2(u', v')$ if $\langle u', v' \rangle \neq \langle v_i, v_j \rangle$.*

Proof. Suppose $C(M_1) \succ C(M_2)$ and there does not exist clause C' such that $C(M_1) \succ C' \succ C(M_2)$. Therefore there exist variables v_i and v_j, $i < j$ such that $C(P_1)\{v_i/v_j\} = C(P_2)$. According to Definition 2 it must be the case that $M_1(v_i, v_j) = 0$ and $M_2(v_i, v_j) = 1$ and $M_1(u', v') = M_2(u', v')$ if $\langle u', v' \rangle \neq \langle v_i, v_j \rangle$. □

Theorem 1. *For each M_1 and M_2 in $\mathcal{M}_n$ if $C(M_1) \succ C(M_2)$ and there does not exist clause C' such that $C(M_1) \succ C' \succ C(M_2)$ then $M_1 \subset M_2$.*

Proof. Suppose $M_1 \not\subset M_2$, thus there exists u' and v' such that $M_1(u', v') = 1$ and $M_2(u', v') = 0$. But according to Lemma 1 there exists unique $\langle v_i, v_j \rangle$, $i < j$ such that $M_1(v_i, v_j) = 0$ and $M_2(v_i, v_j) = 1$ and for each u' and v', $M_1(u', v') = M_2(u', v')$ if $\langle u', v' \rangle \neq \langle v_i, v_j \rangle$. This contradicts the assumption and completes the proof. □

Theorem 2. *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 and only if $M_1 \subset M_2$.*

Proof. $\Rightarrow$: Either there does not exists clause C' such that $C(M_1) \succ C' \succ C(M_2)$ or there exist clauses C_1 and C_2 and $\ldots C_n$ such that $C(M_1) \succ C_1 \succ C_2 \succ \ldots C_n \succ C(M_2)$. In case 1 theorem holds from Theorem 1, in case 2 theorem holds by transitivity.

$\Leftarrow$: 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, $C(P_1)\{v_i/v_j\} = C(P_2)$. Hence, $C(M_1) \succ C(M_2)$. □

Definition 8 (Subsumption Set). *Let B and C both be clauses. C is in subsumption set $\mathcal{S}(B)$ if $C \in \mathcal{B}(B)$ and $M(C) \subset M(B)$.*

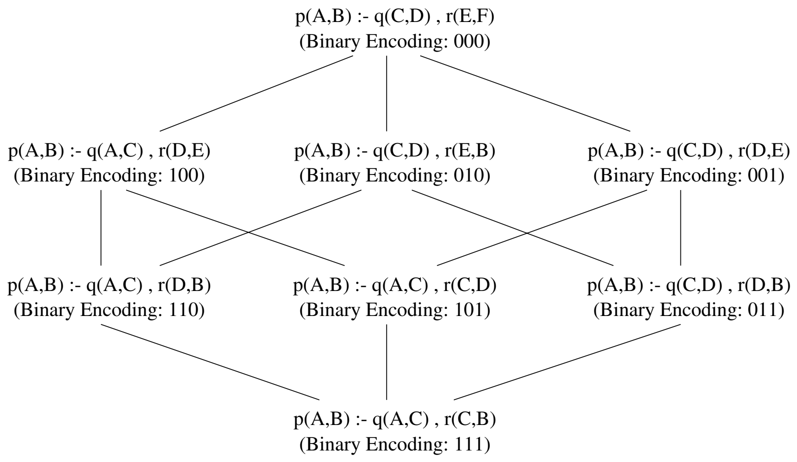

p(A,B) :- q(C,D) , r(E,F)
(Binary Encoding: 000)
p(A,B) :- q(A,C) , r(D,E)
(Binary Encoding: 100)
p(A,B) :- q(C,D) , r(E,B)
(Binary Encoding: 010)
p(A,B) :- q(C,D) , r(D,E)
(Binary Encoding: 001)
p(A,B) :- q(A,C) , r(D,B)
(Binary Encoding: 110)
p(A,B) :- q(A,C) , r(C,D)
(Binary Encoding: 101)
p(A,B) :- q(C,D) , r(D,B)
(Binary Encoding: 011)
p(A,B) :- q(A,C) , r(C,B)
(Binary Encoding: 111)

Fig. 2. Subsumption lattice bounded below by clause *p(A,B):-q(A,C),r(C,B)* and relevant binary encodings.

Theorem 3. *For each clause C and B such that $C \in \mathcal{S}(B)$, $C \succ B$.*

Proof. Suppose $C \in \mathcal{S}(B)$, then $M(C) \subset M(B)$. But according to Theorem 2 $C \succ B$. □

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 4), we are interested in a subsumption lattice bounded below by a particular clause. According to Theorem 3, $\mathcal{S}(B)$ represents a subsumption lattice bounded below by clause B. Each member of $\mathcal{S}(B)$ can be encoded by a bit-string in which each bit corresponds to a 1 entry of the matrix.

Example 1. Figure 2 shows the subsumption lattice bounded below by the clause $p(A, B) : -q(A, C), r(C, B)$. Entries m_{13}, m_{26} and m_{45} of the binding matrix in Figure 1 are encoded by three bits as shown in Figure 2.

3 Genetic Refinement Operators

After a proper representation, well designed genetic operators are important factors for success of a genetic algorithm. Genetic operators introduce new individuals into population by changing or combining the genotype of best-fit individuals during an evolutionary process. In this section we present three genetic operators which can be interpreted in ILP terms if applied on the binary representation introduced in the previous section. These operators are and-operator, or-operator and one-point crossover. We show that these operators are equivalent to some of ILP concepts such as *lgg (least general generalization)* and *mgi(most general instance)* [15].

Definition 9. *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. *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.*

Definition 11. *Let M_1 and M_2 be in $\mathcal{M}_n$ and $1 \leq s_1 \leq n$ and $1 \leq s_2 \leq n$ be randomly selected natural numbers. $M = M_1 \otimes M_2$ is an $n \times n$ matrix and for each $a_{ij} \in M$, $b_{ij} \in M_1$ and $c_{ij} \in M_2$ it is the case that $a_{ij} = b_{ij}$ if $i < s_1$ or $i = s_1$ and $j \leq s_2$ and $a_{ij} = c_{ij}$ otherwise.*

In Definition 11 operator $\otimes$ is equivalent to one-point crossover as defined in genetic algorithms' literatures [4]. Now, we show some interesting properties of these simple operators.

Theorem 4. *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 and only if $M = (M_1 \wedge M_2)$.*

Proof. $\Rightarrow$: Suppose $C(M) = lgg(C(M_1), C(M_2))$. According to the definition of lgg, firstly $C(M) \succ C(M_1)$ and $C(M) \succ C(M_2)$ and according to Theorem 2 and Definition 9 $M \subset (M_1 \wedge M_2)$. Secondly for each binding matrix M' if $C(M') \succ C(M_1)$ and $C(M') \succ C(M_2)$ then $C(M') \succ C(M)$, therefore if $M' \subset (M_1 \wedge M_2)$ then $M' \subset M$. Therefore $(M_1 \wedge M_2) \subset M$ and $M \subset (M_1 \wedge M_2)$, hence $M = M_1 \wedge M_2$.

$\Leftarrow$: Suppose $M = M_1 \wedge M_2$. Therefore $M \subset M_1$ and $M \subset M_2$ and according to Theorem 2 $C(M) \succ C(M_1)$ 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 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 or-operator is equivalent to *mgi* [2].

Theorem 5. *For each clause B and matrices M_1, M_2 and M in $\mathcal{M}_n$ such that $C(M_1) \in L(B)$, $C(M_2) \in L(B)$ and $C(M) \in L(B)$, $C(M) = mgi(C(M_1), C(M_2))$ if and only if $M = M_1 \vee M_2$.*

Proof. Symmetric with proof of Theorem 4. □

[2] The result of operator $\otimes$ can be compared with the result of W operator (predicate invention) in ILP. This property is not discussed in this paper.

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

Because these operators randomly change some entries of the binding matrices, it is possible that the resulting matrix be inconsistent with Definition 2. Such a matrix can be normalized by a closure using Definition 3 before mapping it into a clause.

4 Implementation

In our first attempt, we employed the proposed representation to combine Inverse Entailment in CProgol4.4 with a genetic algorithm. In this implementation genetic search is used for searching the subsumption lattice bounded below by the bottom-clause ($\perp$). According to Theorem 3 the search space bounded by the bottom clause can be represented by $\mathcal{S}(\perp)$. We encoded members of $\mathcal{S}(\perp)$ by bit-strings (as described in section 2) and then applied a genetic algorithm to evolve a randomly generated population of clauses in which each individual corresponds to a member of $\mathcal{S}(\perp)$. Because of simple representation and straightforward operators any standard genetic algorithm can be used for this purpose.

We used a variant of *Simple Genetic Algorithm(SGA)* [4] and modified it to suit the representation introduced in section 2. This genetic search is guided by an *evaluation function* which is similar to one used in A^*-like search of Progol but normalized between 0 and 1. Unlike Progol's refinement operators which non-deterministically decide to include (or exclude) each atom of $\perp$ in current hypothesis, genetic refinement operators evolve bindings of the hypothesis in $\mathcal{S}(\perp)$. Atoms which violate the mode declaration language [13], defined by the user, are filtered from each clause before evaluation. The details for Progol's refinement operator, algorithm for building $\perp$ and A^*-like search can be found in [13] and the algorithm and other details of *Simple Genetic Algorithm(SGA)* can be found in [4]. In the first experiment, we applied the genetic search to learn Michlaski's trains problem [11]. 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). In this experiment, the following setting was used for *SGA* parameters: $popsize = 30$, $p_c = 0.6$ and $p_m = 0.0333$.

Preliminary results show that the standard Progol A^*-like search exhibits better performance in learning hypothesis with small and medium complexities. However, the performance of genetic search is less dependent on the complexity of hypothesis, 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 A^*-like search due to its incompleteness [13, 1].

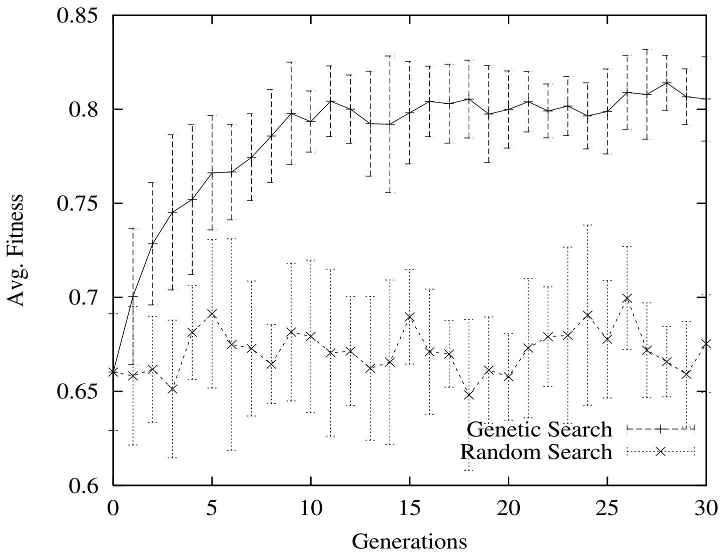


Fig. 3. Convergence of the genetic search in the trains problem.

5 Discussion and Related Works

The actual completeness and complexity exhibited by the standard A^*-like search of Progol depends upon the order of atoms in the bottom clause and upon the distance of the acceptable hypothesis from the root of the search tree. In contrast, because of a multi-point strategy it uses, the genetic algorithm is more regular search method and is not affected by the order of atoms 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 hypothesis into bit-strings. GA-SMART [3] and later REGAL [2] and DOGMA [5] were relation learning systems which tackled this problem by restricting concept description language and introducing language templates. A template, in GA-SMART, is a fixed length CNF formula which must be obtained from domain theory (or 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. In REGAL and DOGMA a template is a conjunction of internally disjunctive predicates which introduce a more complicated language, but still incomplete and poor for representing some features (e.g. continuous attributes). Other systems including GILP [16], GLPS [10] and STEPS [7] use hierarchical representations rather than using fixed length bit-strings. These systems evolve a population of logic programs in a Genetic Programming (GP) [9] manner. Most of the above mentioned systems cannot use any form of intensional background knowledge and a few of them use limited forms of background knowledge just for generating initial population or seeding the population.

On the other hand, in our proposed framework, binary representation of hypothesis is compact and complete in comparison to all other methods which use a binary encoding. Encoding of solutions is based on a bottom clause constructed according to the background knowledge using some ILP 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 ILP concepts. Hence, in terms of genetic algorithms theory, the proposed framework is not only consistent with *the principal of minimal alphabets* in GAs but also it is along the right lines of *the principle of meaningful building blocks* [4].

6 Conclusions and Further Work

In this paper we have introduced a framework for combining GAs with ILP methods. Simple but complete and compact binary representation for encoding clauses and ILP interpretations of this representation (and relevant operators) are the major novelty of the proposed framework.

An implementation of this framework is used to combine Inverse Entailment of CProgol with a genetic search. Even though this implementation justifies the properness of the proposed framework, it could be improved in many ways. An improvement might be using more sophisticated genetic algorithms (e.g. distributed genetic algorithms) rather than using a simple genetic algorithm. More experiments are also required in complex domains and noisy domains. The ILP interpretation of the proposed 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 ILP systems. As genetic refinement operators introduced in this paper concern specialization as well as generalization, the proposed framework could be useful for theory revision as well.

Undoubtedly, GAs and ILP are on opposite sides in the classification of learning processes [8]. While GAs are known as empirical or BK-poor, ILP 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.

Acknowledgements

Alireza Tamaddoni-Nezhad would like to thank his wife, Parisa Nejabati, for her support during the writing of this paper. Many thanks are also due to the three anonymous referees for their comments, Suresh Manandhar for discussions on the related topics and Alen Varšek for providing us a summary of [16].

References

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

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

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

[4] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA, 1989.

[5] J. Hekanaho. Dogma: A ga-based relational learner. In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 205–214. Springer-Verlag, 1998.

[6] J.H. Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.

[7] Claire J. Kennedy and Christophe Giraud-Carrier. 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, April 1999.

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

[9] J. R. Koza. *Genetic Programming.* MIT Press, Cambridge, MA, 1991.

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

[11] R.S. Michalski. Pattern recognition as rule-guided inductive inference. In *Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 349–361, 1980.

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

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

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

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

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