Learning from Noise Data with the Help of Logic Programming Systems

ELENA BAUER
Department of Programming Systems
Friedrich-Alexander University Erlangen-Nuremberg
Martensstrasse 3, 91058 Erlangen
GERMANY

GABRIELLA KÓKAI
Department of Programming Systems
Friedrich-Alexander University Erlangen-Nuremberg
Martensstrasse 3, 91058 Erlangen
GERMANY

www2.informatik.uni-erlangen.de/~kokai

Abstract: - This paper presents an overview of the recent systems, that combine inductive logic programming with genetic algorithms. The systems are described and then compared to their design and their behaviour with correct and incorrect training data.

Key-Words: - Inductive Logic Programming, Evolutionary Algorithms, Genetic Algorithms, Genetic Programming, Concept Learning, Relational Learning, First Order Logic

1 Introduction

Evolutionary algorithms (EA) are stochastic general purpose search-algorithms, that have been applied to a wide range of machine learning problems. EAs can not learn recursive structures, but they have the advantage that the whole hypothesis space is searched through.

Learning from examples in first-order logic, also known as inductive logic programming (ILP), is also a central topic in machine learning. ILP can learn recursive structures, but ILP has the disadvantage that the hypthesis space is only partially searched through.

In 1994 Wigham and McKay had the idea to combine EAs and ILP to utilise the advantages of both approaches: The ability of genetic algorithms to search a wide search-space efficiently and the ability of ILP to learn recursive structures. In this paper five genetic logic programming (GLP) systems are described. They are compared first to their physical structure and then to their performance on correct and incorrect training data.

This paper is organized as follows. Section 2 introduces evolutionary algorithms and inductive logic programming. Section 3 describes five systems which combine EAs and ILP. Section 4 compares the systems to their design and section 5 to their accuracy. Finally section 6 summarizes the results and gives an outlook to future work.

2 Theoretical Background

This section gives a briefly introduction to the two learning methods EA and ILP which are used by the five systems that are to be discussed.

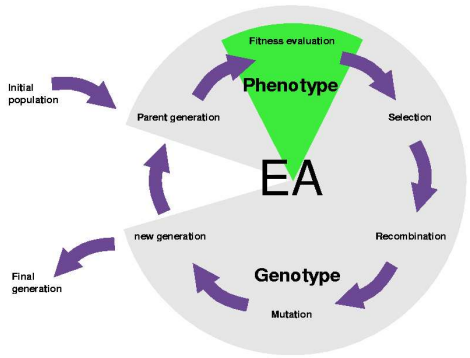

Figure 1: The evolution process

2.1 Evolutionary Algorithms

Evolutionary algorithms (EA) [7] are population-based learning algorithms and they can deal with comparatively difficult optimisation-problems. EAs offer an efficiently search-algorithm to a lot of learning tasks. The cycle of an EA is shown in figure 1.

Genetic algorithms (GA) and genetic programming (GP) are partitions of EAs.

2.2 Inductive Logic Programming

Inductive logic programming (ILP) is a machine learning approach, in which correlations of objects are ascertained by induction. Hypotheses are searched for and evaluated by comparing their classification results with a sufficiently large number of instances for which knowledge exists, whether their objects are correlated

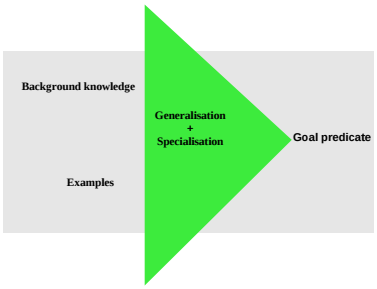

Figure 2: General construction of ILP

or not [11]. It is thus assumed that hypotheses classifying these training instances correctly will also approximate the target function well over any other set of instances. The learned hypotheses can be interpreted as Prolog programs, since they consist of a set of rules, that is, first order Horn clauses.

The most used operators (see Figure 2) with which *consistent* and *complete* programs can be generated are *specializing* and *generalizing*. A hypothesis H is consistent, if H covers no negative example. H is complete, if it covers all positive examples. In addition to consistency and completeness there is a third feature to evaluate a hypothesis: *simplicity*, which results in shorter hypotheses.

Specializing is a learning method where the required starting hypothesis stipulates that it must terminate for all positive and some negative examples. Result is a modified starting hypothesis. The actual hypothesis is specializied as long as negative examples are no longer derivable. *Generalizing* can easily be regarded as the opposite of specializing. Here the process does not terminate for some positive examples. Frequently the starting hypothesis is the empty program or the program in which the errors are to be found.

3 Introduction of the GLP-systems

In this section the GLP-systems are described by their data representation, their architecture, their operators (selection, mutation, crossover, seeding) and their fitness functions.

3.1 REGAL

RElational Genetic Algorithm based Learner (REGAL, [6]) is a distributed GA-based learner, which learns first-order logic concept descriptions from examples.

The concept description language, used by REGAL, is a Horn clause language. A predicate for example has the form $predicate(x_1, x_2, ..., x_m, K)$. x_1, x_2, ..., x_m are variables and the term K is a disjunction of constants. One single individual is a conjunction of predicates and is encoded as a bitstring. A *language template* defines the order and complexity of the resulting bitstring.

In REGAL one single process, called the *supervisor*, coordinates a set of so called *Nodal Genetic Algorithms* (NGAs). Each of these NGAs searches a partition of hypothesis space given by the supervisor and sends its best results back frequently. The supervisor extracts a temporary solution from these results. If this is a satisfactory solution, the supervisor stops the NGAs and finishes the search, otherwise the supervisor can focus the search to other parts of the hypothesis space by modifying the set of the examples assigned to each NGA.

REGAL uses a special operator to select individuals within the genetic algorithm. This operator is called the *Universal Suffrage* (US) operator. A number of positive training examples are randomly selected. All the individuals that cover one of these selected examples are determined. One individual is selected with the probability proportional to its fitness. The selected individual is then replaced with its offspring.

REGAL has four different crossover operators for reproduction. For further explanation of the classical *uniform* or classical *two-point* crossover see [7]. The specializing and generalizing operator works by flipping bits. The more bits in a bitstring are set the more general is the bitstring and vice versa. If the generalizing operator is chosen, new substrings are generated by OR-ing the bits of randomly selected corresponding substrings of the parents. Substrings that are not selected are copied unchanged into the new individuals. The specializing operator works analogously with a logical AND.

The mutation operator inverts every bit in the bitstring with the same probability.

The seeding operator gets as parameter a positive instance and returns an individual which covers this instance. This operator is used while initializing and as mutation operator.

The fitness function depends on completeness, consistency and simplicity.

3.2 G-NET

G-NET [1], a succesor of REGAL has the same concept description language and for this reason the same

data representation as REGAL.

The architecture of G-NET encompasses the three nodes:

- Genetic nodes (*G-nodes*), where individuals are reproduced,
- Evaluation nodes (*E-nodes*), where individulas are evaluated, and
- *Supervisor* node, which coordinates the computation of the G-nodes according to a *coevolutive strategy* [1].

The main improvement lies in the coevolutive strategy. The selection operator is the US operator as described for REGAL.

The crossover operators used by G-NET are a classical two-point crossover [7] and a variant of the uniform crossover, which performs either specializing or generalizing. The generalizing and specializing operators work just as the ones of REGAL.

The mutation operator is either specializing, generalizing or seeding. The specializing mutation operator is performed by turning zeros into ones. The generalizing operator by turning ones into zeros.

G-NET has two different fitness functions, one to evaluate the global (disjunctive) and one to evaluate the local (conjunctive) fitness. These functions are based on completeness, consistency and simplicity.

3.3 DOGMA

The concept description language and for this reason the data representation of Domain Oriented Genetic MAchine (DOGMA, [8]) is similar to the language and data representation of REGAL described above.

DOGMA operates on two different levels: The lower and the upper level. The lower level is the bitstring-based level of REGAL with appropriate operators. The crossover, mutation and seeding operators are the same as in REGAL, additionally there are a *mate* and a *background-seeding* operator. In the upper level the individuals are divided into *genetic families* with special family operators: A *break*, a *join* and a *make-family* operator is implemented.

DOGMA uses background knowledge: Individuals are divided into *species* according to which part of background knowledge they may use. In one family there can be only one individual of the same species. This helps to enhance diversity in the population.

A next generation is build in DOGMA by applying the different operators (both genetic and family) successively.

The fitness function of DOGMA is a combination of the minimum description length principle (MDL) and information gain (IG).

3.4 ECL

Evolutionary Concept Learner (ECL, [4]) computes if-then-rules of the form $p(X, Y) \leftarrow r(X, Y), q(Y, a)$. The rules consists of atoms; the arguments of the atoms are either variables or constants.

ECL loops the following algorithms: First a part of the background knowledge is chosen. A number N of individuals are selected and these individuals are reproduced. The loop will end, if either all positive examples are covered or a maximum number of iterations are processed. In the second case a logic program is extracted which covers as much positive and as less negative examples as possible.

ECL has three different selection operators: The US operator of REGAL, described in section 3.1 and two extensions of this operator: the *Weighted Universal Suffrage* (WUS) operator and the *Exponentially Weighted Universal Suffrage* (EWUS) operator. These two operators favor examples more difficult to deal with, thus examples that are uncovered or examples that are covered only by few clauses.

ECL does not use crossover operators, but it has two specializing and two generalizing mutation operators: A clause is generalized either by deleting an atom from the body of a clause or by replacing all occurrences of a constant in a clause with a variable. A clause is specialized either by adding an atom to the body or by replacing a variable with a constant. Every mutation operator uses a gain function, which yields the difference between the clause fitness before and after the application of the mutation operator: If for example the *AtomDeletion* operator is choosen, the atom at in the clause which omission yields to the highest gain is deleted.

The fitness function of ECL is based on consistency and completeness.

3.5 GELOG

The GELOG [10] framework is another GLP framework, that combines ILP with an evolutionary search algorithm. Especially, GELOG's data representation resembles the one of GP: The individual solutions consist of Prolog program parts, which encode the hypotheses' rules. Thus, an individual comprises the target predicate as its left hand side and a number of disjunctions (right hand sides), which are conjunctions of literals. The following example demonstrates how in-

dividuals are represented in the the original GELOG implementation:

```
daughter(X0, X1) :-
     female(X0), parents(X1, X2, X0).
     parents(X2, X1, X0).
     female(X1), female(X0),
     parents(X2, X1, X0).
```

The depicted individual consists of three disjunctions (right hand sides); each disjunction contains a number of conjuncted literals and is terminated by a dot.

The pay-off of one hypothesis results from the number of correctly classified instances. Different selection operators have been implemented: fitness proportional selection, rank based selection and elitism (for further explanation of these operators see [7, 9]).

Due to the non-standard data representation, special recombination and mutation operators had to be implemented:

- Recombination operators:
  - two individuals exchange whole right hand sides by single- or multi-point crossover,
  - two individuals exchange literals of their right sides by performing single- or multi-point crossover.
- Mutation operators:
  - insertion and deletion of literals,
  - insertion and deletion of entire disjunctions,
  - insertion of new variables, and
  - substitution of variables.

The fitness function is based on the consistency, completeness and the simplicity of the hypothesis.

Considering the size of the class of problems GELOG can be applied to, it was inevitable that the limits of the system quickly became obvious and under certain circumstances or for certain problems, the system thus turned out to be less efficient. Consequently, works were realized, where it was tried to broaden the existing system and thus resolve some of the surging problems by using methods such as meta evolution, parameter adaptation, seeding or linkage learning.

4 Comparison of the GLP systems

After the representation of the systems, they are compared to their most important features: data representation, operators, fitness functions and parallelization.

4.1 Data representation

The data representation of the GLP systems is shown in table 1. DOGMA, G-NET and REGAL use a bitwise representation for the individuals and use language templates to map the attributes to a bitstring. ECL and GELOG use high-level data representation.

GLP System	data representation
REGAL	bitstring, needs language template
DOGMA	bitstring, needs language template
G-NET	bitstring, needs language template
ECL	high level language representation
GELOG	logic program

Table 1: Data representation

A good choice seems to be a high level representation like ECL or GELOG, because the individuals can have variable length and the specializing and generalizing operators can be applied directly and may not lead to inconsistent individuals.

4.2 Genetic Operators

Table 2 shows the comparison of the operators of the different systems.

Mutation and Crossover

The high level representation of GELOG and ECL leads to almost identical mutation operators. Both systems have operators to insert or delete a new literal and to insert or substitute variables. GELOG additionally uses crossover operators.

Selection

The two extensions of the US operator in ECL are a better solution than the pristine US operator. The US operator does not distingiush between examples that are harder to cover and examples that are easier to cover. This can lead to an emergence of so called *superindividuals*. These individuals cover many (easy) examples. These superindividuals will often be selected and the diversity of the population will decrease. The selection operators of ECL are better than the selection operators of REGAL and G-NET.

DOGMA uses a simple fitness proportional selection operator, which can also lead to superindividuals. GELOG uses in addition to the fitness proportional selection operator, an elitism and a rank based selection operator. The rank selection operator prevents the occurrence of superindividuals.

The difference between GELOG and ECL relies on the fact that in GELOG the selection occurs over the whole population and in ECL the selection occurs over a part of the population (over the individuals that are selected randomly by the selection operator).

GLP System	mutation operators	crossover operators	selection operators
REGAL	classical seeding	uniform, two-point generalizing, specializing	Universal Suffrage
DOGMA	classical seeding	uniform, two-point generalizing, specializing	fitness proportional on family-level
G-NET	generalizing, specializing, seeding	two-point generalizing, specializing	Universal Suffrage
ECL	2 specializing 2 generalizing	none	Universal Suffrage, WUS, EWUS
GELOG	7 different (specializing, generalizing, neither)	2 RecombineRHSOperators 2 RecombineTwoRHS- PredicateOperators	fitness proportional elitism, rank

Table 2: Comparison of the operators

4.3 Fitness

The differences of the fitness functions are described in table 3.

GLP System	fitness function
REGAL	consistency, completeness, simplicity
DOGMA	Minimum Description Length, Information Gain
G-NET	2 functions: consistency, simplicity, completeness
ECL	consisteny, completeness
GELOG	consistency, completeness, simplicity

Table 3: fitness functions

The fitness function of DOGMA is based on completeness and consistency, because the MDL implemented in DOGMA preferres fairly large, but very inconsistent rules and the IG preferres almost consistent, but very incomplete rules.

Therefore there are no great differences in the fitness functions: all systems use completeness and consistency to evaluate the fitness of an individual.

The feature simplicity is missed in ECL and in DOGMA. Although it is not such an important feature like completeness and consistency, shorter hypotheses are more simple to handle, to save and to verify.

4.4 Parallelization

REGAL and G-NET are per se parallel algorithms, the focus of the search can be shifted to promising parts of the hypothesis space. DOGMA and GELOG have the ability to parallelize the evolution process, DOGMA has a *Parallel Virtual Machine* architecture and GELOG has an extension of a so called *meta evolution*.

5 Accuracy

In this section the systems are compared using three types of datasets: The mushroom and the chess datasets are taken from the *UCI Machine Learning Repository* [2] and the mutagenesis dataset is taken from [3].

Unlike DOGMA for all systems the 10-fold cross validation was used to get the average accuracy. DOGMA was trained on five small training sets, each containing 100 examples and the accuracy was then tested on 5000 randomly generated examples. The results of ECL and DOGMA are from [4] and [8]. The results of GELOG, G-NET and REGAL were tested by the authors.

In table 4 the average accuracy obtained by the systems on different datasets is presented.

GLP System	Chess	Mushrooms	Mutagenesis
REGAL	–	99.7	–
DOGMA	97.31	–	–
G-NET	96.34	100.0	86.1
ECL	–	–	87.2
GELOG	99.32	99.75	–

Table 4: Result is the average accuracy obtained by the systems on different datasets

GELOG shows good results for the chess and the mushroom dataset. But the best results on the mushroom dataset has G-NET.

In table 5 DOGMA and GELOG are compared with the ILP system mFOIL [5]. mFoil was developed to handle noisy datasets; the results are not directly comparable with the results of DOGMA and GELOG, because mFoil does not use 10-fold cross validation. The

NL	arguments			arguments and class			class		
%	mFOIL	DOGMA	GELOG	mFOIL	DOGMA	GELOG	mFOIL	DOGMA	GELOG
5	91.87	**94.36**	85.48	89.28	**92.75**	84.89	**94.26**	92.47	84.85
10	80.45	**88.70**	84.88	80.79	80.23	**84.86**	**92.02**	90.64	84.15
15	80.06	83.37	**84.91**	76.58	78.07	**84.82**	**89.96**	88.86	84.97
20	76.49	80.86	**84.74**	74.74	74.16	**84.89**	88.37	**88.77**	84.52
30	74.04	72.72	**84.69**	69.65	67.08	**84.80**	86.01	**87.14**	84.60
50	68.62	61.84	**84.89**	66.27	56.20	**84.79**	**79.20**	77.13	57.12
80	66.30	56.42	**74.98**	**61.73**	52.85	42.65	**65.04**	53.72	36.35

Table 5: Results of the chess problem. NL refers to level of noise.

comparison is based on corrupted training data. Three kinds of noise were considered: noise only in the class variable, noise in the arguments and noise in the arguments and in the class variable.

GELOG produces stable results for almost all noise levels. The system accuracy is about 85%. DOGMA and mFoil have very good results for the 5% error level and for the training data in which the classvariables are corrupted.

6 Conclusion and Future Work

This paper described five GLP systems. Every system combines GA with ILP.

The prime similiarity of REGAL, G-NET and DOGMA lies in their bitwise data representation. Especially for the chess problem DOGMA proves efficient on less corrupted training data.

The expected benefits from GELOG's high-level data representation are demonstrated by its stable results on noisy training data. The comparison between GELOG and ECL was not possible because we do not have information about ECL's behaviour on corrupted training data.

In the future we plan further comparisons, with a broader range of datasets and with a simple genetic algorithm.

References:

[1] C. Anglano, A. Giordana, G. Lo Bello, and L. Saitta. An Experimental Evaluation of Coevolutive Concept Learning. In *Proceeding of the 15th International Conference on Machine Learning*, pages 19–27, 1998.

[2] C. Blake and C. Merz. UCI Repository of machine learning databases.

[3] A Debnath, R. L. de Compadre, G. Debnath, A. Schusterman, and C. Hansch. Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medical Chemistry*, 34(2):786–797, 1991.

[4] F. Divina and E. Marchiori. Evolutionary Concept Learning. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 343–350, 2002.

[5] S. Džeroski. Handling imperfect data in inductive logic programming. In *Proceedings of the 4th Scandinavian Conference on Artificial Intelligence*, pages 111–125. IOS Press, 1993.

[6] A. Giordana and F. Neri. Search-Intensive Concept Induction. *Evolutionary Computation Journal*, 3(4):375–416, 1996.

[7] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[8] J. Hekanaho. DOGMA: A GA-based Relational Learner. In *Proceedings of the 8th International Workshop on Inductive Logic Programming*, pages 205–214, 1998.

[9] C. Jacob. *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann, 2001.

[10] G. Kókai. GeLog - a System Combining Genetic Algorithm with Inductive Logic Programming. In *Proceedings of the International Conference on Computational Intelligence*, pages 326–345, 2001.

[11] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.