Learning first order logic rules with a genetic algorithm

S. Augier[1], G. Venturini[1,2] and Y. Kodratoff[1]

[1]Equipe Inférence et Apprentissage
Laboratoire de Recherche en Informatique
Bat. 490, Université de Paris-Sud
91405 Orsay Cedex FRANCE
augier, yk@lri.fr

[2]Laboratoire d'Informatique
E3i, Université de Tours
64, Avenue Jean Portalis, Technopôle Boîte No 4
37913 Tours Cedex 9 FRANCE
venturi@lri.fr

Abstract

This paper introduces a new algorithm called SIAO1 for learning first order logic rules with genetic algorithms. SIAO1 uses the covering principle developed in AQ where seed examples are generalized into rules using however a genetic search, as initially introduced in the SIA algorithm for attribute-based representation. The genetic algorithm uses a high level representation for learning rules in first order logic and may deal with numerical data as well as background knowledge such as hierarchies over the predicates or tree structured values. The genetic operators may for instance change a predicate into a more general one according to background knowledge, or change a constant into a variable. The evaluation function may take into account user preference biases.

Introduction

The so-called supervised learning framework is the main way to apply Machine Learning (ML) techniques to rule discovery. In such a framework, a set of examples is collected, where each example is of the form "description → class". Each observation (or case) is thus labelled by a class. The supervised learning algorithm must find informative regularities in the observations to explain their classes, and suggest these regularities to the user.

In supervised learning, the most common representation for describing the examples is the attribute/value representation which uses propositional logic. Each example is described with a finite set of attributes (or variables) that may take either numeric or symbolic values. This representation has the advantage of being shared by numerous knowledge discovery techniques like for instance statistics, neural networks, data analysis, decision trees or genetic algorithms. The same database (DB) can thus be analyzed by several tools. However, this representation has the drawback of being unable to represent relations between entities in the DB. Relational DBs often express relations among fields that are equivalent to subsets of FOL. Thus, using first order logic (FOL) to describe the examples is a way to be able to handle this aspect of the DB. Most of the ML algorithms that deal with first order representation use deterministic and heuristic techniques for discovering knowledge, like for instance FOIL (Quinlan & Cameron-Jones 1993). They may get trapped in local optima.

Genetic algorithms (GAs) are adaptive procedures that evolve a population of chromosomes structures in order to find the fittest individual. This evolution is performed by a selection operator and two genetic operators namely mutation and crossover. Since optimization techniques based on GAs are less sensitive to local optima, using such algorithms for learning FOL rules seems to be a good idea. However, one major obstacle is to find how to represent such rules as a chromosome and to define genetic operators for modifying their genes. The crossover operator requires preferably a fixed size representation, and this requirement does not fit directly the FOL representation where examples may be described with a variable number of predicates.

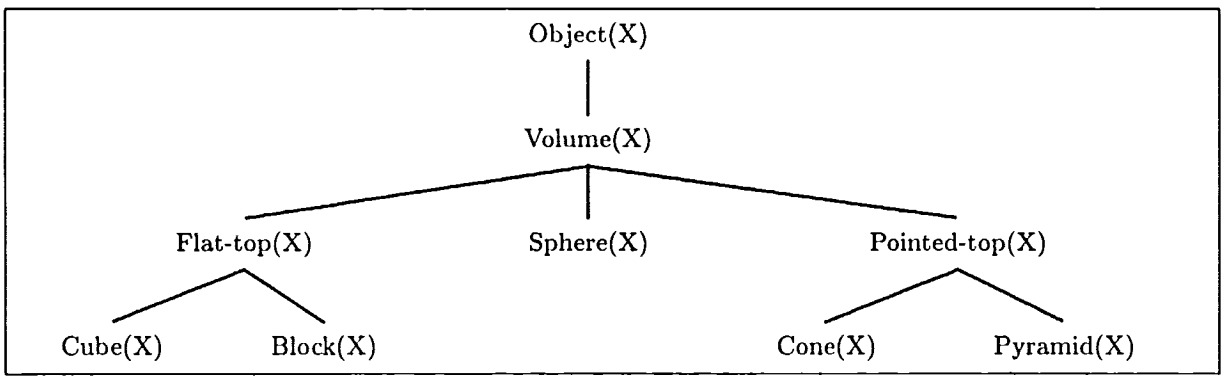
This paper introduces a new algorithm $\mathcal{SIAO}1$ that is a generalization of SIA, a supervised learning system that uses attribute/value representation only and not FOL (Venturini 1994). REGAL (Giordana & Saitta, 1993) (Giordana, Saitta & Zini, 1994) is the only known solution for learning rules in FOL with GAs. To deal with this representation obstacle, REGAL assumes that the user can provide a general model, called a template, of the formula to be learned, which we do not. In this paper, we shall have no place to compare with REGAL. Let us notice, however, that REGAL can also induce more complex formulas than $\mathcal{SIAO}1$. Section 3 will compare the two approaches with more details.

Problem to be solved

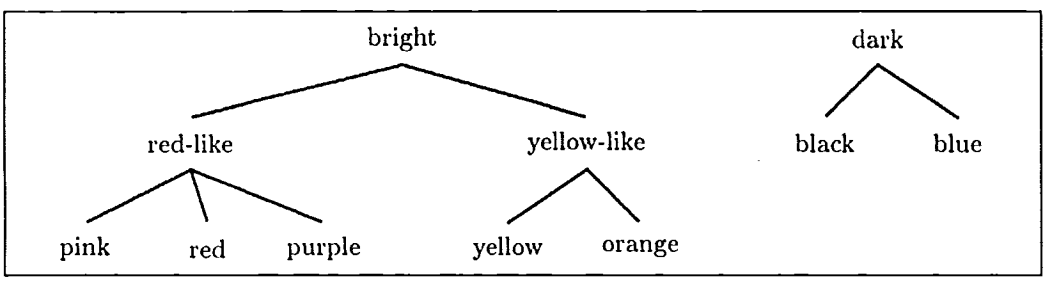
The general problem considered in this paper is to discover rules in FOL from a set of examples and background knowledge. The examples are described with a conjunction of predicates without quantifiers or function symbols and may belong to two classes, positive or negative, as shown in figure 1(a). The predicates may represent symbolic information like the predicate Color(X,value) as well as numeric information like

Description part		Class
Cube(a) ∧ Block(b) ∧ Cone(c) ∧ Color(a,yellow) ∧ Color(b,pink) ∧ Color(c,red) ∧ Length(a,11) ∧ Length(b,11) ∧ Length(c,18) ∧ Supports(a,c) ∧ Supports(b,c)	→	⊕
Object(d) ∧ Cube(e) ∧ Cone(f) ∧ Color(d,purple) ∧ Color(e,blue) ∧ Color(f,orange) ∧ Length(d,6) ∧ Length(e,7) ∧ Length(f,8) ∧ Supports(e,d)	→	⊖

(a) A positive and a negative example.



(b) A possible hierarchy of predicates.



(c) A possible hierarchy of the values of a predicate ("value" in Color(X,value) for instance).

Figure 1: A simple example of a database and of background knowledge $\mathcal{SIAO}1$ deals with.

the predicate Length(X,value). Background knowledge can be represented in at least two ways: some predicates may be organized into a generalization hierarchy (figure 1(b)) and the values of a symbolic predicate may be also organized in the same way (see figure 1(c)).

From this set of examples and from background knowledge, the learning system should discover regularities that could explain why an example belongs to class $\oplus$ for instance. This can be achieved by learning rules like :

$$\begin{array}{c}\text{Flat-top(X)}\wedge\text{Flat-top(Y)}\wedge\text{Pointed-top(Z)}\wedge\\ \text{Color(X,bright)}\wedge\text{Color(Y,bright)}\wedge\text{Color(Z,bright)}\wedge\\ \text{Length(X,L)}\wedge\text{Length(Y,L)}\wedge\text{Length(Z,[1,20])}\wedge\\ \text{Supports(X,Z)}\wedge\text{Supports(Y,Z)} \quad\rightarrow\quad \oplus\end{array}$$

which represent the concept of an arch. Obviously the representational power of FOL is needed for discovering such a regularity because a relation between entities is involved.

Representing first order logic rules in a genetic algorithm

A genetic algorithm usually works with individuals encoded as binary strings with fixed length. One bit represents the possible value of a gene, and a string represents a chromosome which is the genetic encoding of an individual. Genetic operators that modify chromosomes using this representation are the mutation and the crossover. Mutation modifies some bits, and crossover exchanges substrings between two parents. The aim of this last operator is to combine useful information in order to produce better offsprings.

GAs are interesting for solving ML problems because, as mentioned in the introduction, they make less assumptions about the objective function than heuristic methods, and can be also parallelized. However, the crossover operator is constraining the kind of possible representations for learning rules because rules must be represented as a string of genes. A rule in propositional logic is already a string of elements because it always uses the same number of attributes. This is the reason why most of GAs applications in rule learning use an attribute-based representation (Holland 1986) (De Jong 1988) (De Jong & Spears 1991). For FOL representation, the problem is to find a good representation, i.e, a representation that would let the crossover play its role, and that could represent interesting rules for the user.

REGAL is a successful attempt to deal with FOL representation in GAs (Giordana & Saita 1993) (Giordana, Saitta & Zini 1994). REGAL requires that the user provides a model of the rule to be learned. For instance, the user may provide the following model (Giordana & Saitta 1993) :

$$\begin{array}{c}\text{Color(X,[red,blue,*])}\wedge\text{Shape(X,[square,triangle,*])}\wedge\\ \neg\exists\ \text{Y [color(Y,[red,blue,*])}\wedge\text{far(X,Y,[0,1,2,3,*])]}\end{array}$$

Then, the GA is used to discover the correct values in the predicates. A possible set of values can thus be encoded as a string of genes, and can be represented as a binary string. For instance, the values of "Shape" can be represented by a 3 bits string. For a predicate with numeric values, each bit encodes one possible value. An additional bit may be used to encode the negation of the internal disjunction of values.

On one hand, REGAL has several advantages from the GA point of view: the GA works directly with binary strings of fixed length, as required by the original GA theory. This theory has however been extended to other representation languages (Radcliffe 1991) with n-ary alphabets instead of binary. REGAL is also able to encode complex formulas in the initial model with for instance quantifiers or negation, because the GA does not work on quantifiers but only on sets of values. Finally, the model provided by the user can be viewed as background knowledge.

On the other hand, REGAL has several limitations from the point of view of knowledge discovery. Most importantly, the user must provide the structure for the rules to be learned. This assumes that the user has already a rough idea of what he expects to discover. Also, REGAL does not deal with the kind of background knowledge mentioned in figure 1. It does not modify the variables in the predicates. Finally, for dealing with numeric values, the binary representation of the model may be very long which may experimentally slow down the GA. For instance, if one considers that the Length predicate can take 100 values, the binary encoding will be 101 bits long (including the internal disjunction).

This paper provides a new technique for representing FOL rules which overcomes the previously mentioned limitations. One first difference is that rules will be represented directly in FOL by using the predicates and their arguments as genes. The GA will be allowed to perform other operations than in REGAL, like for instance changing a predicate into a more general one according to the background knowledge, or like changing a constant into a variable. However, two genes at the same position in two rules must represent the same predicate, if the crossover is to work well. This problem is solved by using a covering principle in the rule discovery process: an example is used as an initial model and the GA is used to discover the best generalization of this example. Then another example is chosen as a new model in order to learn another rule until all examples are covered by learned rules. This covering process comes from the AQ algorithm (Michalski et al. 1986). The combination of this covering algorithm with a GA was firstly introduced in (Venturini 1994) with the SIA algorithm. SIA is however limited to attribute-based representation and SIAO1 can be viewed as a major extension of SIA.

SIAO1

A covering principle

$\mathcal{SIAO}1$ main algorithm is a covering algorithm similar to AQ (Michalski et al., 1986). The system tries to find from an example ex the best rule r according to a rule-evaluation criterion taking into account the coherence of the rule, its completeness and other features like the syntactic generality of the rule and the user's preferences for some predicates. Then all the positive examples covered by r are removed, and the system applies the same process to an example ex' chosen from the remaining set of the non-covered positive examples. This results in the following algorithm, where $\mathcal{R}$ is a disjunct of each learned rule:

Star methodology

1. $\mathcal{R} \leftarrow \emptyset$,
2. $\mathcal{P}os \leftarrow$ Examples of the concept to be learned.
3. $\mathcal{N}eg \leftarrow$ Counter-examples of the concept.
4. While $\mathcal{P}os \neq \emptyset$ do
   (a) Choose $ex \in \mathcal{P}os$ as a seed.
   (b) Find the best rule r (according to a rule-evaluation criterion) obtained by generalization from ex.
   (c) Remove from $\mathcal{P}os$ all the examples covered by r.
   (d) $\mathcal{R} \leftarrow \mathcal{R} \cup r$.
5. Give $\mathcal{R}$ as the result.

The optimization algorithm used in the previous point 4.(b) is a GA, which is called each time a seed example must be generalized into a rule. The search space is thus limited to the possible generalizations of the seed example. This example is used to determine the model of the rule to be learned, and is thus not given by the user as in REGAL. Furthermore, with another seed example, this model will be different and is not constant as in REGAL.

SIAO1 search algorithm

The search process begins with only one individual (the seed) in the population denoted by Pop. To obtain the new generation, the algorithm applies to each individual a genetic operator (mutation or crossover), and inserts possibly the newly created individual in Pop. The size of the population grows until it reaches a bound given by the user (typically $\mathcal{P}max = 50$). The algorithm is shown below:

1. $Pop \leftarrow$ { Seed }
2. Repeat
   (a) Generate an intermediate population: for every individual r of Pop, if r has not already produced any offsprings, then create:
      i. one offspring by mutation with probability P_m $r \rightarrow r'$
      ii. two offsprings by crossover with probability P_c $r, r' \rightarrow r_1, r_2$. The other parent r' is selected randomly in the population among rules that have not produced yet any offsprings.
   (b) For every individual r of the intermediate population, possibly insert r in Pop only if $r \notin Pop$ and either:
      - the maximum size of the population is not reached (r is added to Pop), or
      - the maximum size of the population is reached, and r fitness is better than the worst fitness observed in Pop (r replaces the worst individual).
3. Until the fitness of the best individual observed has not changed since the last n generations, and then give this best individual as the result.

The default parameters are $P_m = 0.8, P_c = 1 - P_m = 0.2$ and $n = 5$. This algorithm has the following properties:

- two individuals in the population are necessarily different. This maintains in the population the diversity required to prevent premature convergence that may occur in GAs. The difference guaranteed between individuals is syntactic, and it proved to be a good alternative in terms of complexity to the use of more complex similarity measures, which are difficult to define in first order logic,
- the stopping of the algorithm is ensured in the following way: the best individual in the population is never deleted due to the selection operator and it may only be improved by the genetic operators. Thus the fitness of the best individual is strictly increasing and is also bounded,
- the search algorithm is non-deterministic, and in particular $\mathcal{SIAO}1$ is sensitive to the choice of the seeds which will be processed.

Genetic operators

The genetic operators have been adapted to the covering principle of the algorithm. Thus the mutation, operating on a gene generalizes it, and the crossover is a standard one-point crossover which exchanges information between two individuals. Let us consider for instance the following rule r:

$$\text{Pyramid(d)} \wedge \text{Color(d,yellow)} \wedge \text{Supports(c,d)} \wedge \text{Length(d,7)} \rightarrow \oplus$$

encoded as the string of genes represented in figure 2.

More precisely, the mutation selects randomly a relevant gene on which it will operate randomly one of the following modifications:

- if the gene codes a predicate, it is possible to generalize it according to the domain theory, for instance "Pyramid" could be changed into "Pointed-top" without modifying the arguments of the predicate. If no more generalization of the predicate is possible, then the mutation will drop it, replacing

Pyramid	d	Color	d	yellow	Supports	c	d	Length	d	7

Figure 2: The chromosome coding rule r.

the former symbol by an "empty" symbol, meaning that the predicate and the arguments immediately following it are not to be considered anymore. The corresponding part of the chromosome becomes a non-coding part, and no mutation will take place there (having no effects).

- if the gene codes a numeric constant, then the mutation can create an interval, or if such an interval already exists, the mutation can widen it. The following operations are thus possible: "7" → "[7,9]", "[7,9]" → "[6,9]". The bounds of the intervals are generated randomly in a user given range,
- if the gene codes a symbolic constant, then the mutation may create an internal disjunction or generalize an existing disjunction. This may result in the following operations: "yellow" → "yellow ∨ orange" , "yellow ∨ orange" → "yellow ∨ orange ∨ blue". A value like "yellow" may also be changed to "yellow-like" according to the background knowledge,
- it is also possible to change a numeric or symbolic constant into a new variable. Because creating new variables is interesting as long as relations can be detected, this kind of mutation may have an effect not only on the gene it is destined to, but on the entire chromosome: it finds all the occurrences of the symbol concerned in the chromosome and then randomly replaces them by the new variable or leaves them unchanged. A mutation operating on the second "d" of r could generate for instance one of the following rules:
"Pyramid(d)∧Color(X,yellow)∧Supports(c,d)∧...",
"Pyramid(X)∧Color(X,yellow)∧Supports(c,d)∧...",
"Pyramid(d)∧Color(X,yellow)∧Supports(c,X)∧...",
"Pyramid(X)∧Color(X,yellow)∧Supports(c,X)∧...",
assuming that "X" is a new variable occurring nowhere else in the formula.
- finally the mutation can apply the same process of variabilization to a variable instead of a symbolic or numeric constant.

Two crossover operators are available. The first one (used by default) randomly determines a cut point located before a predicate and exchanges the two remaining strings (see figure 3). The offsprings that result from this restrained crossover operator are always valid because they are generalizations of the same seed example. This restrained one-point crossover is useful because it prevents the appearance of the hidden arguments following an empty predicate. Allowing a cut point just after the "Pyramid" predicate would on the previous example hide the "X" variable in the first offspring and reveal the argument hidden due to the empty predicate. Since this kind of behavior must be avoided, it has been impeded.

If the seed contains only one predicate symbol, the restrained one-point crossover becomes useless, and therefore the classical one-point crossover applies. This is the case when one wants to use $\mathcal{SIAO}1$ on a classical attribute/value learning base. In such a case it is possible to add the same arbitrary predicate symbol before each line of data, with no domain theory for the predicates.

Evaluation function

The fitness function gives a numerical evaluation of an individual. In $\mathcal{SIAO}1$ like in many genetic algorithms, this function has been empirically designed, and gives a result in [0,1]. Four criterions are taken into account:

- the *consistency* of the rule. Because the algorithm proceeds mainly by generalization, this criterion is critical: a rule covering too many negative examples will continuously generate inconsistent rules. Therefore, the fitness function will strongly penalize any inconsistent rule, and it will give a score of 0 to any rule covering more counter-examples than allowed by a noise parameter,
- the *completeness* of the rule. Unlike consistency, this criterion gives a positive contribution to the fitness, growing linearly with the number of positive examples covered,
- the *syntactic generality* of the rule computed with a home-made formula returning a value in [0,1]; this allows the algorithm to favor between rules with equal completeness over the set of examples, the one containing the more variables and the largest disjunctions and intervals. This criterion is particularly important in the case of a learning base whose examples are distant -from a mutation point of view- in the search space.
- the *user's preferences.* In $\mathcal{SIAO}1$, the user can fix parameters to favor the presence of a given predicate in a rule, or to favor the presence of relations between predicate arguments. Fulfilling the user's preferences is expressed by another function which gives a result in [0,1].

Given $E\oplus$ the number of examples covered, $E\ominus$ the number of counter-examples covered, $T\oplus$ the total number of examples, $T\ominus$ the total number of counter-examples, the maximum noise tolerated $\mathcal{N}$,the syntactic generality $\mathcal{S}$ of the rule and it's appropriateness to the user's preferences $\mathcal{P}$, we can define:

- the absolute consistency of the rule $\mathcal{CN} = \frac{T\ominus - E\ominus}{T\ominus}$,

(a)

Pyramid	X	Color	X	yellow	Supports	Y	X	∅	∅	∅
∅	∅	Color	Y	yellow	Supports	c	d	Length	d	7

(b)

Pyramid	X	Color	X	yellow	Supports	c	d	Length	d	7
∅	∅	Color	Y	yellow	Supports	Y	X	∅	∅	∅

Figure 3: The crossover applied to two parents rules (a) with a single cut point (denoted by "||") may give two offsprings rules (b).

- its absolute completeness $\mathcal{CM} = \frac{E\oplus}{T\oplus}$

Therefore, the quality of an individual r will be:

- $(1-\alpha-\beta)\mathcal{CM} + \alpha\mathcal{S} + \beta\mathcal{P}$ if $\mathcal{CN} > 1 - \mathcal{N}$
- 0 otherwise.

α and β are user-defined parameters, but typically $\alpha < 10.\beta$ and $0 \leq \alpha+\beta < \frac{1}{T\oplus+T\ominus}$ meaning that user's preferences are more important than the absolute generality of a formula, and that these two criterions cannot prevent preference for a more complete rule.

Conclusion

We have presented in this paper a new learning algorithm based on genetic algorithms that learns first order logic rules from a set of positive and negative examples and from background knowledge. $\mathcal{SIAO}1$ represents rules in a high level language, and is thus able to perform high level operations such as generalizing a predicate according to the background knowledge or like changing a constant into a variable.

Genetic algorithms thus prove that they can represent and learn rules in high level language involving relations, as initially demonstrated by REGAL, and now by $\mathcal{SIAO}1$. In this context, the ability of these algorithms to be parallelized is certainly essential compared to other heuristic techniques. (Giordana, Saitta & Zini, 1994)

Current research on $\mathcal{SIAO}1$ concern the parallelization of the algorithm, its application to real DBs and its extension to deal with more expressive and complex domain theories.

References

De Jong K. 1988. Learning with Genetic Algorithms: An overview. *Machine Learning* 3, 121-138.

De Jong K., and Spears W.M. 1991. Learning concept classification rules using genetic algorithms, Proceedings of the 12^{th} International Joint Conference on Artificial Intelligence 1991, J. Mylopoulos and R. Reiter (Eds), 651-656, Morgan Kaufmann.

Goldberg D.E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*: Addison Wesley.

Giordana A., and Saitta L. 1993. Regal: an integrated system for learning relations using genetic algorithms, Proceedings of the Second International Workshop on Multistrategy Learning 1993, R.S. Michalski et G. Tecuci (Eds), pp 234-249.

Giordana A., Saitta L., and Zini F. 1994. Learning disjunctive concepts by means of genetic algorithms, Proceedings of the 1994, Proceedings of the Eleventh International Conference on Machine Learning, 96-104.

Holland J.H. 1975. *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

Holland J.H. 1986. Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems, Machine Learning: an AI approach, volume 3, R.S. Michalski, T.M. Mitchell, J.G. Carbonell et Y. Kodratoff (Eds), Morgan Kaufmann, 593-623.

Michalski R.S. 1983. Theory and methodology of inductive learning, Machine Learning: An AI Approach, Volume 1, R.S. Michalski, T.M. Mitchell, J.G. Carbonell et Y. Kodratoff (Eds), Morgan Kaufmann,

Michalski R.S., Mozetic I., Hong J., and Lavrac N. 1986. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains, Proceedings of AAAI-86 Fifth National Conference on Artificial Intelligence, Morgan Kaufmann, 1041-1045.

Quinlan J.R., and Cameron-Jones R.M. 1993. FOIL: a midterm report, Proceedings of the European Conference on Machine Learning 1993, P. Brazdil (Ed.), Lecture notes in artificial intelligence 667, Springer-Verlag, 3-20.

Radcliffe N.J. 1991. Equivalence class analysis of genetic algorithms. *Complex systems*, 5(2), 183-205.

Venturini G. 1994. Analyzing French Justice with a genetic based inductive algorithm, *Applied Artificial Intelligence*, R. Trappl (Ed), Special Issue on Real world applications of Machine Learning, Y. Kodratoff (Guest Ed), vol 8, N 4, 565-577.

Winston P. 1975. Learning structural descriptions from examples, *Psychology of computer vision*, Mc Graw-Hill, NY, 1975.