UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

Learning Constraints between Variables in Inductive Logic Programming

Agnès Braud, Christel Vrain
Université d'Orléans, LIFO

Rapport N^o **2001-06**

Abstract

In Inductive Logic Programming, a critical point is the complexity of the algorithms inherent in first order representations. An interesting solution to this, called propositionalization, consists in reformulating a relational problem into an attribute-value one. A pattern describing the general form of the rules is given or learned, and the system has to specialize it by learning either symbolic constraints, as for instance equality relations between variables, or numeric ones.

We propose to use a genetic approach to specialize such patterns. In this paper, we are interested in learning suitable equality constraints between variables. The main originality of our approach relies on the coding of these constraints and the corresponding genetic operators. Instead of coding in an individual whether pairs of variables are equal or not, we represent the equivalence classes involved by the equality relations, and apply set-based operators on them to produce new individuals.

1 Introduction

We are interested in the application of Inductive Logic Programming (ILP) techniques to Data Mining (DM) and we mainly focus on the classification task that can be defined as learning an intentional definition of a concept from examples stored in a database. Many works in Data Mining assume that all data are stored in a single relation and thus require relational databases to be flattened in a single relation, loosing thus important information on the structure of the database and leading to a huge and often intractable relation.

In such a context, one of the main advantages of ILP for Data Mining is its ability to handle relational data involving several relations. On the other hand, a critical point for ILP is the complexity arising in first order representations. This complexity arises mostly from the size of the search space and depends on the number of relations, their arities and when necessary on the introduction of function symbols. Several approaches have been developed to deal with this problem, as for instance the definition of biases [10]. An interesting solution to this complexity problem, called propositionalization, consists in reformulating a relational problem into an attribute-value one. It has already been studied in some systems, as for instance LINUS [8], STILL [11], REPART [13], PROPAL [1]. A pattern describing the general form of the rules to be learned is given by the user or learned, and the system has to specialize it by learning either symbolic constraints, as for instance equality relations between variables, or numeric ones. To illustrate this, let us consider a very simple example and let us suppose that we want to learn the concept of $grand\_father$ providing positive and negative examples of $grand\_father$, and two background relations $father$ and $mother$. If the user guesses that a definition of $grand\_father$ involves the two relations $mother$ and $father$ and that they occur only once in the definition, a possible pattern could then be:

$$grand\_father(X_1, X_2) \leftarrow mother(X_3, X_4), father(X_5, X_6),$$

thus letting the system learn that to get a correct definition of $grand\_father$, the symbolic relations $X_5 = X_1$, $X_2 = X_4$ and $X_3 = X_6$ must hold. A more complicated pattern including more predicates could also be considered, as for instance:

$$grand\_father(X_1, X_2) \leftarrow father(X_3, X_4), father(X_5, X_6), mother(X_7, X_8).$$

In that case, several solutions can be found:

- $X_1 = X_3$, $X_2 = X_8$, $X_4 = X_7$, leading to the clause
$grand\_father(X_1, X_2) \leftarrow father(X_1, X_4), father(X_5, X_6), mother(X_4, X_2)$.

  We can notice that the variables X_5 and X_6 are not linked to variables occurring in the head of the clause, and therefore that the literal $father(X_5, X_6)$ can be removed.

- $X_1 = X_5$, $X_2 = X_8$, $X_6 = X_7$, leading to
$grand\_father(X_1, X_2) \leftarrow father(X_3, X_4), father(X_1, X_6), mother(X_6, X_2)$.

- $X_1 = X_3 = X_5$, $X_2 = X_8$, and $X_4 = X_6 = X_7$, leading to
$grand\_father(X_1, X_2) \leftarrow father(X_1, X_4), father(X_1, X_4), mother(X_4, X_2)$.

  that can be simplified by removing one of the two literals $father(X_1, X_4)$.

- $X_1 = X_3$, $X_2 = X_6$, and $X_4 = X_5$, leading to
$grand\_father(X_1, X_2) \leftarrow father(X_1, X_4), father(X_4, X_2), mother(X_7, X_8)$.

Let us notice that even on the first example involving 6 variables, 15 equalities between pairs of variables must be studied. This can somewhat be reduced when variables are typed, allowing only equalities between variables with the same type (here man/woman). Moreover, other kinds of constraints can be interesting as for instance numeric constraints.

In this paper, we focus only on learning equality constraints between pairs of variables and we investigate the use of Genetic Algorithms (GAs) that allow a stochastic search in the hypotheses space. This problem has also been addressed in [12] and formalized as searching the subsumption lattice. In their approach, links between variables are coded by a binary matrix where a bit indexed by i and j represents whether $X_i = X_j$ or not, and the evolutionary process is applied to learn the best ones, that is those which implies the best rule. For this purpose, operators on matrixes have been defined to simulate one-point crossover, specialization and generalization of the rule. The drawback of this method is that the number of possible equalities increases rapidly with the number of variables appearing in the starting pattern. Moreover some genes in an individual do not add any information, as for example when we have three genes representing respectively $X = Y$, $Y = Z$ and $X = Z$: if two of them are set to 1, the value of the third can only be equal to 1.

In this paper, we propose another approach which consists in coding in an individual the equivalence classes suggested by the equality relations. For instance, the relations $X_1 = X_3 = X_5$, $X_2 = X_8$, and $X_4 = X_6 = X_7$ can be represented by three classes, namely $\{X_1, X_3, X_5\}$, $\{X_2, X_8\}$, and $\{X_4, X_6, X_7\}$. The problem can then be stated as follows: given a set of variables, find an optimal partition of them.

This paper is organized as follows. In Sect. 2, we present the genetic approach used to solve the problem of learning a partition of a set of objects, the coding and the operators we propose. Then, in Sect. 3 we show how this can be used in an ILP context, to learn symbolic constraints between variables in a pattern. Finally, in Sect. 4, we compare our work with other related ones before concluding in Sect. 5.

2 Learning a Partition of a Set of Objects

Definition of the problem:
Given
- a finite set of objects $\mathcal{O} = \{o_1, \ldots, o_n\}$,
- a criterion for evaluating a partition of $\mathcal{O}$,

Find a partition of $\mathcal{O}$ optimal for this criterion.

Let us recall that a partition $\mathcal{P}$ defines an equivalence relation $\mathcal{R}$ by
$$\mathcal{R}(o_i, o_j) \iff o_i \text{ and } o_j \text{ belong to the same class in } \mathcal{P}.$$
Thus learning $\mathcal{P}$ is the same task as learning $\mathcal{R}$.

This problem, known as clustering, induces a search space with a size that grows very quickly with the number of objects. The number of partitions of a set of n objects is given by the $(n+1)^{th}$ Bell number (the sequence begins with the number of partitions for the empty set). To give an idea of the tendency of this sequence, its first numbers are the following: 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, ... Due to their great explorative power, GAs are appealing for this task. Several genetic approaches have thus deal with the problem of clustering ([3, 7] for example), but most of the time they actually dealt with k-clustering, and in this case, the number k of classes of the partition to learn is given beforehand. This imposes a strong bias on the search space and considerably reduces the size of the search space. However, it is hard to determine k *a priori*.

Let us consider a set $\mathcal{O}$ of 6 objects of the same type, and a partition $\mathcal{P}$ of these objects into three classes $\{1,6\}$, $\{2,3\}$ and $\{4,5\}$. Some of the genetic approaches consider the partition as a graph [3], where each object is a node and two nodes belonging to a same cluster are linked by an edge. It is then possible to use a binary encoding, where each bit corresponds to a possible edge and is set to 1 if this edge exists, to 0 otherwise. In a graph composed of 6 nodes, there are 15 possible edges, and $\mathcal{P}$ would then be represented by the individual 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0.

Some other approaches represent a partition by a permutation of the objects [3, 7]. The clusters are indicated in the individuals by separators ($\mathcal{P}$ can then be represented by the individual 1 6 | 2 3 | 4 5), or determined dynamically when evaluating the individual using a given heuristic: the k first numbers are the indexes of the objects that are the seeds for the k clusters, and the other objects are allocated to the cluster which maximizes a criterion, one after the other in the order they appear. Genetic operators designed for permuting genes are suitable for this representation.

Another kind of representation is called group-number encoding [7]. In this representation, a partition of a set of n objects is a string of integers of length n, where the i^{th} integer represents the class of the i^{th} object. Thus, the individual 1 2 2 3 3 1 is a representation of $\mathcal{P}$. The problem is that the label of a cluster can be meaningless, so that genetic operators can be misleading if no renumbering is done. For example, let us consider the individuals 1 2 2 3 3 1 and 1 3 3 2 2 1. They encode the same partition; nevertheless, if we apply the traditional one-point crossover on them, and we choose a cut point between the third and fourth genes, the children (1 2 2 2 2 1 and 1 3 3 3 3 1) will encode new partitions with only two clusters, while we would have expected to obtain the same as in the parents.

In the following, we describe a genetic approach for solving clustering, without any restriction on the number of clusters to learn. The representation used is based on the group-number encoding, that we have somewhat modified, by adding a constraint on the labelling of the clusters. This allows to prevent from the drawbacks of this encoding. We also propose set-based operators.

2.1 Individuals Encoding

Let $\mathcal{O}$ be a set of n objects that are numbered $o_1, \ldots, o_n$.

An individual encodes a partition of $\mathcal{O}$ as described below.

Each equivalence class of the partition is given a label. An individual contains as many genes as there are objects in $\mathcal{O}$. It can be seen as an array Ind, indexed from 1 to n, where the value $Ind[i], i \in \{1, \ldots, n\}$, is the label of the class containing o_i.

For labelling a class, we consider the objects it contains and we take the lowest index of these objects. For instance, if a class contains $\{o_3, o_4, o_6\}$, its label will be equal to 3. This latter rule ensures uniqueness in the representation of a partition, this will also be helpful to implement genetic operators.

Proposition 1 *The representation of a partition by an individual is unique.*

Proof 1 *Let us denote by Ind_1 and Ind_2 two individuals encoding the same partition $\mathcal{P}$ of $\mathcal{O}$.*

Let us suppose $Ind_1 \neq Ind_2$, that is there exists $i \in \{1, \ldots, n\}$ such that $Ind_1[i] \neq Ind_2[i]$.

Let us set $Ind_1[i] = j$ and $Ind_2[i] = k$. We denote by $\mathcal{C}_{1,j}$ the class labelled j in Ind_1 and $\mathcal{C}_{2,k}$ the class labelled k in Ind_2. Thus, $o_i \in \mathcal{C}_{1,j}$ in Ind_1 and $o_i \in \mathcal{C}_{2,k}$ in Ind_2, and since Ind_1 and Ind_2 encode the same partition of $\mathcal{O}$, $\mathcal{C}_{1,j} = \mathcal{C}_{2,k}$.

Moreover, by construction of Ind_1 (a class is labelled by the index of its object of smallest index), the object of smallest index in $\mathcal{C}_{1,j}$ is o_j; similarly the object of smallest index in $\mathcal{C}_{2,k}$ is o_k.

Since $\mathcal{C}_{1,j} = \mathcal{C}_{2,k}$, and j and k are the smallest indexes in these classes, we thus have $o_j = o_k$ and $j = k$, that contradicts the hypothesis.

Figure 1 shows an example of the representation of a partition of a set of objects by an individual.

set of objects	$\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5, o_6\}$
coding pattern	o_1 o_2 o_3 o_4 o_5 o_6 | | | | | | |
a potential partition of the objects	$\mathcal{C}_1 = \{o_1, o_5\}$, $\mathcal{C}_2 = \{o_2\}$, $\mathcal{C}_3 = \{o_3, o_4, o_6\}$
the corresponding individual	1 | 2 | 3 | 3 | 1 | 3

Figure 1: An example of coding

2.2 Set-based Genetic Operators

Individuals that better split the set of objects into equivalence classes will be given best qualities and higher chance to reproduce. Our genetic operators have to allow to propagate good equality relations from parents to children, and to introduce new ones to preserve diversity in the population. As some distant genes can belong to the same class and thus be highly linked in an individual, we cannot use the traditional blind crossover operators, such as n-points crossover, that exchange parts of two individuals and are very sensitive to the ordering of genes. Even uniform crossover would not be suitable. Indeed, it would enable to exchange many genes involving many modifications in the partition. This would be too disruptive if individuals are not similar enough to allow an efficient exploitation of information contained in parents, and it would impose many treatments to preserve the consistency of the individuals. Actually, it seems to be a bad strategy to work on more than one equivalence class in each individual during a genetic operation applied on two individuals. We therefore use some operators, more intelligent although still stochastic, that simulate traditional operations encounter in set theory.

We first give a general description of these operators, then we will explain how we apply them in a propositionalization framework and discuss their effects when applied in this context in Sect. 3.

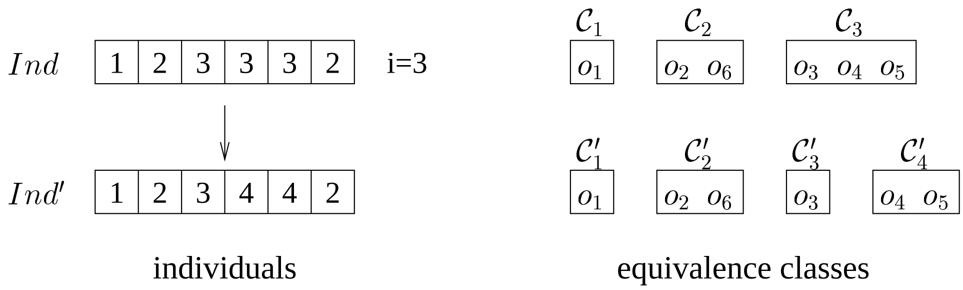

Figure 2: Isolating an object

2.2.1 Operators Involving One Individual

Let us denote by Ind the individual concerned, by Ind' its child. For each operator, we define the modifications applied to Ind to generate its child.

On the figures, we will denote by $\mathcal{C}_l$ a class labelled l in Ind, by $\mathcal{C}'_l$ a class labelled l in Ind'.

Isolating an Object. This operator consists in putting an object o_i, $o_i \in \mathcal{C}$, alone in a new equivalence class. We randomly select i, then three cases can be encountered:

- o_i is already isolated, then nothing is done;
- Class $\mathcal{C}$ is labelled l ($Ind[i] = l$) with $l < i$, then $Ind'[i]$ is set to i, and for all the other genes k of Ind', $Ind'[k] = Ind[k]$;
- Class $\mathcal{C}$ is labelled i (the object index), then the other objects that belong to $\mathcal{C}$ are put in a new class, by setting their corresponding genes to j, where j is the index of the object of lowest index among them. For all $k \in \{1, \ldots, n\}$ such that $Ind[k] = i, k \neq i$, $Ind'[k] = j$ where $j = \min(k)$, $k \neq i$ and $Ind[k] = i$. For all the others $k \in \{1, \ldots, n\}$, $Ind'[k] = Ind[k]$.

Figure 2 shows an example where the operator is applied on the object o_3. As it belongs to a class labelled by its index ($Ind[3] = 3$), Gene 3 keeps its value in Ind' and the genes of the other objects of $\mathcal{C}_3$ are set to 4, as o_4 has the lowest index among them.

Let us denote by P the initial partition, by P' the partition of the child. Let us set $P = Q + \mathcal{C}$, the effects of this operator can be mathematically formulated by $P' = Q + \{\mathcal{C} - o_i\} + \{o_i\}$.

Moving an Object across Classes. An object o_i is extracted from its class $\mathcal{C}_{l_1}$ and added to a class $\mathcal{C}_{l_2}$. Three cases are possible:

- $l_1 = l_2$, o_i is already in Class $\mathcal{C}_{l_2}$, then nothing is done;
- $l_1 \neq l_2$ and $i > l_2$, then $Ind'[i]$ is set to l_2. Moreover, if $l_1 = i$ (i is the smallest index in $\mathcal{C}_{l_1}$ and is used as label for the class) then for all k such that $o_k \in \mathcal{C}_{l_1}$ and $k \neq i$, $Ind'[k]$ is set to m where m is the smallest of these indexes;
- $l_1 \neq l_2$ and $i < l_1$, then all the genes corresponding to objects of $\mathcal{C}_{l_1}$ are set to i in Ind'. Again, if $l_1 = i$ then for all k such that $o_k \in \mathcal{C}_{l_1}$ and $k \neq i$, $Ind'[k]$ is set to m where m is the smallest of these indexes.

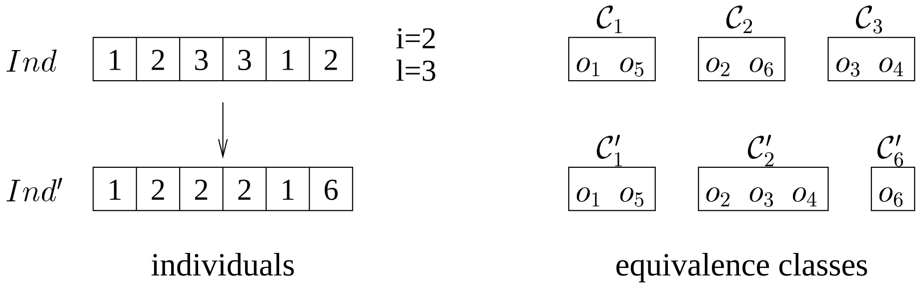

Figure 3: Moving an object across classes

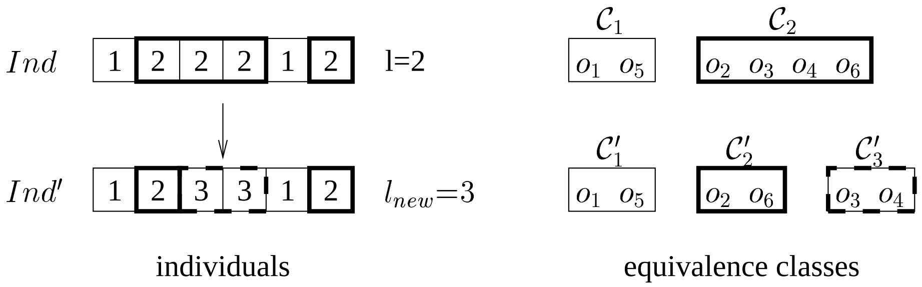

Figure 4: Splitting a class into two ones

For all the other genes j of Ind', $Ind'[j] = Ind[j]$.

On Fig. 3, we can see an example, where the object indexed 2, belonging to the class $\mathcal{C}_2 = \{o_2, o_6\}$ in Ind, is chosen to be added to the class $\mathcal{C}_3 = \{o_3, o_4\}$. We are thus in the third case: the genes equal to 3 in Ind are set to 2 in Ind', and as o_6 was in $\mathcal{C}_2$ in Ind, Gene 6 is set to 6 not to include it in the new class.

Let us denote by $\mathcal{C}_{init}$ the class from which we extract o_i. Let us set $P = Q + \mathcal{C}_{init} + \mathcal{C}_l$, the operator transforms P into $P' = Q + \{\mathcal{C}_{init} - o_i\} + \{\mathcal{C}_l + o_i\}$.

Splitting a Class into Two Ones. A class $\mathcal{C}_l$ of Ind is split into two ones. For that purpose, we randomly select a class index l among the labels of the classes in Ind. Then, we randomly select some genes among those equal to l in Ind, except Gene l, and set them to k in Ind', where k is the smallest index among them. For all the other genes i of Ind', $Ind'[i] = Ind[i]$.

For example, on Fig. 4, the class $\mathcal{C} = \mathcal{C}_2 = \{o_2, o_3, o_4, o_6\}$ is split. A new one, labelled $l_{new} = 3$, appears in Ind'; this class contains two objects randomly selected among those of Class $\mathcal{C}_2$ in Ind, and o_3 has the smallest index among them.

Thus, the partition $P = Q + \{\mathcal{C}_i + \mathcal{C}_j\}$ is transformed into $P' = Q + \mathcal{C}_i + \mathcal{C}_j$.

Let us notice that isolating an object is a special case of splitting a class into two ones. Nevertheless, the operation that isolate an object often cuts less links between objects in $\mathcal{C}$, so that it can be more interesting when few changes are necessary in the partition to get a better solution.

Merging Two Classes. Two classes are merged into a unique one. First, we randomly select two classes $\mathcal{C}_{l_1}$ et $\mathcal{C}_{l_2}$. Let us suppose $l_1 < l_2$; then for each $i \in \{1, \ldots, n\}$ such that $Ind[i] = l_2$, $Ind'[i]$ is set to l_1. For all the other genes i of Ind', $Ind'[i] = Ind[i]$.

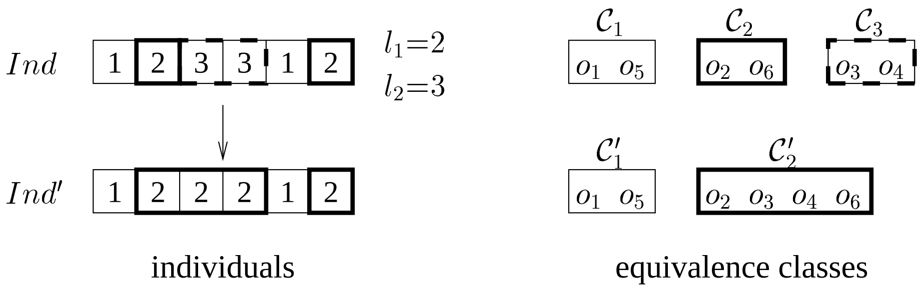

Figure 5: Merging two classes

On Fig. 5, Classes $\mathcal{C}_2$ et $\mathcal{C}_3$ are selected in Ind, then their objects are gathered into a single class labelled 2 in Ind' (their corresponding genes are set to 2).

This operator thus transforms a partition $P = Q+\mathcal{C}_i+\mathcal{C}_j$ into $P' = Q+\{\mathcal{C}_i+\mathcal{C}_j\}$.

2.2.2 Operators Involving Two Individuals

Let us denote by Ind_1 and Ind_2, the two individuals on which this operator is applied, by Ind'_1 and Ind'_2 their children. Once again, we describe how to modify Ind_1 and Ind_2 to create their offspring.

On the figures, we will denote by $\mathcal{C}_{i,l}$ a class labelled l in Ind_i, by $\mathcal{C}'_{i,l}$ a class labelled l in Ind'_i

Union. A class $\mathcal{C}_{1,l_1}$ is selected in Ind_1, and another one $\mathcal{C}_{2,l_2}$ in Ind_2 such that $\mathcal{C}_{1,l_1}$ and $\mathcal{C}_{2,l_2}$ have at least one common object. After the application of the operator, Ind'_1 and Ind'_2 encode the same partition as Ind_1 and Ind_2 respectively, except that variables of $\mathcal{C}_{1,l_1}$ et $\mathcal{C}_{2,l_2}$ are gathered into a same class $\mathcal{C}'_{union}$.

We realize that as follows. If $l_1 \leq l_2$ (resp. $l_2 < l_1$), then for all $i \in \{1,\ldots,n\}$ such that $Ind_1[i] = l_1$, $Ind'_2[i]$ is set to l_1 (resp. l_2) and if $Ind_2[i] = l_1$, $Ind'_1[i]$ is set to l_1 (resp. l_2).

It may be necessary to change the values of genes corresponding to objects from other classes, not to include them in the resulting class, as in the following example. For all the other genes k of Ind'_1 (resp. Ind'_2), $Ind'_1[k] = Ind_1[k]$ (resp. $Ind'_2[k] = Ind_2[k]$).

On Fig. 6, the class labelled $l_1 = 2$ has been selected in Ind_1. It contains o_2 and o_5, therefore this label appears at 2 and 5 in Ind_1. Only the classes of o_2 and o_5 can be chosen in Ind_2 (the ones labelled 2 and 4, as $Ind_2[2] = 2$ and $Ind_2[5] = 4$), in order to have an object in common between the classes selected in the parents. Class $\mathcal{C}_{2,2}$ is chosen. Then the value 2 is set in the Ind'_1 and Ind'_2, for all positions i where either $Ind_1[i] = 2$ or $Ind_2[i] = 2$. The resulting children have all the objects of the starting classes in the same class and only those ones. Let us note that Genes 3 and 6 have been set to 3 in Ind'_2, not to include objects o_3 and o_6 in the resulting class.

Let us denote by P_1 and P_2 the partitions coded by Ind_1 and Ind_2 respectively, P'_1 and P'_2 the partitions coded by Ind'_1 and Ind'_2. Let us set $P_1 = Q_1 + \mathcal{C}_{1,l_1} + \{\mathcal{C}_{1,i} \mid \mathcal{C}_{1,i} \cap \mathcal{C}_{2,l_2} \neq \emptyset \ and\ i \neq l_1\}$ and $P_2 = Q_2 + \mathcal{C}_{2,l_2} + \{\mathcal{C}_{2,i} \mid \mathcal{C}_{2,i} \cap \mathcal{C}_{1,l_1} \neq \emptyset \ and\ i \neq l_2\}$, then $P'_1 = Q_1 + (\mathcal{C}_{1,l_1} \cup \mathcal{C}_{2,l_2}) + \{\mathcal{C}_{1,i} - (\mathcal{C}_{1,i} \cap \mathcal{C}_{2,l_2}) \mid \mathcal{C}_{1,i} \cap \mathcal{C}_{2,l_2} \neq \emptyset \ and\ i \neq l_1\}$ and $P'_2 = Q_2 + (\mathcal{C}_{1,l_1} \cup \mathcal{C}_{2,l_2}) + \{\mathcal{C}_{2,i} - (\mathcal{C}_{2,i} \cap \mathcal{C}_{1,l_1}) \mid \mathcal{C}_{2,i} \cap \mathcal{C}_{1,l_1} \neq \emptyset \ and\ i \neq l_2\}$

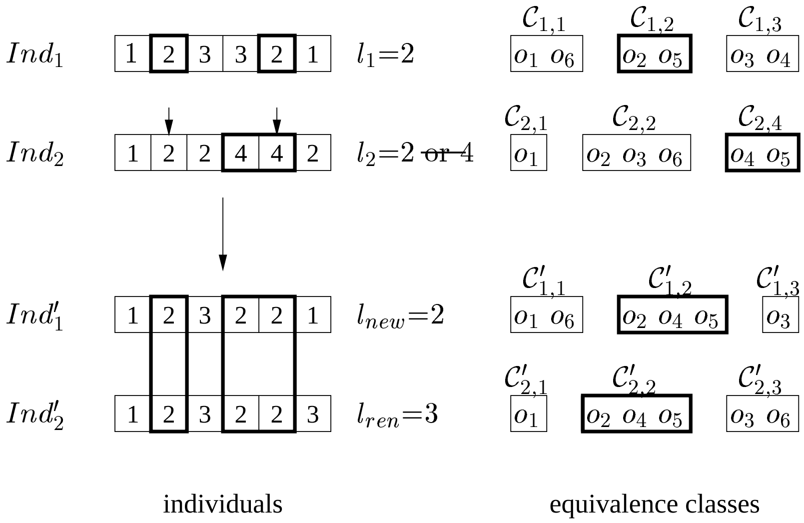

Figure 6: Union

Intersection. A class $\mathcal{C}_{1,l_1}$ is selected in Ind_1, another $\mathcal{C}_{2,l_2}$ in Ind_2 such that $\mathcal{C}_{1,l_1}$ and $\mathcal{C}_{2,l_2}$ have at least an object in common. After the application of the operator, Ind'_1 and Ind'_2 code the same partition as Ind_1 and Ind_2 respectively, except that variables common to $\mathcal{C}_{1,l_1}$ and $\mathcal{C}_{2,l_2}$ are gathered into a class $\mathcal{C}'_{inter}$ that appears in both individuals.

The process is the following. If $l_1 \leq l_2$ (resp. $l_2 < l_1$), then for all $i \in \{1, \ldots, n\}$ such that $Ind_1[i] = l_1$ and $Ind_2[i] = l_2$, $Ind'_1[i]$ and $Ind'_2[i]$ are set to l_1 (resp. l_2). Moreover, for all $i \in \{1, \ldots, n\}$ such that $Ind_1[i] = l_1$ and $Ind_2[i] \neq l_2$, $Ind_1[i]$ is set to j, where j is the smallest index among $\mathcal{C}_{1,i} - \mathcal{C}_{2,j}$. This is necessary to exclude these objects from the resulting class.

As for union, it may also be necessary to change the values of some genes from other classes not to include them in the resulting class.

For all other genes k of Ind'_1 (resp. Ind'_2), $Ind'_1[i] = Ind_1[i]$ (resp. $Ind'_2[i] = Ind_2[i]$).

For example, on Fig. 7, the class labelled $l_1 = 2$ has been selected in Ind_1, and the class labelled $l_2 = 2$ in Ind_2, so that objects o_2 and o_6 are both common to the two classes. In the resulting children, o_2 and o_6 are effectively in the same class and are the only one in it. Let us note that genes at positions 2 and 3 have been set to 3 in the first individual, not to include objects o_3 and o_4 in the resulting class.

Let us denote by $P_1 = Q_1 + \mathcal{C}_{1,l_1}$ and $P_2 = Q_2 + \mathcal{C}_{2,l_2}$, then $P'_1 = Q_1 + (\mathcal{C}_{1,l_1} \cap \mathcal{C}_{2,l_2}) + (\mathcal{C}_{1,l_1} - \mathcal{C}_{1,l_1} \cap \mathcal{C}_{2,l_2})$, and $P'_2 = Q_2 + (\mathcal{C}_{1,l_1} \cap \mathcal{C}_{2,l_2}) + (\mathcal{C}_{2,l_2} - \mathcal{C}_{1,l_1} \cap \mathcal{C}_{2,l_2})$.

For these two operators, we require a common object between the two classes involved. Since these classes belong to individuals selected for their good quality, the equality relations induced by these classes are potentially interesting for the problem, and we can expect to gather interesting links connected by this object. Moreover, in the case of intersection, this condition is necessary to have an effect. In the case of union, this prevents from having too many side effects. For example, if we had chosen Classes $\mathcal{C}_{1,1}$ in Ind_1 and $\mathcal{C}_{2,4}$ in Ind_2 for the operation of Fig. 6, each class would have been modified to get the children, with nothing indicating, neither in Ind_1 nor in Ind_2, that it can be valuable to gather o_1, o_6 and o_4, o_5.

Let us note that in the case of non-homogeneous types for the objects in $\mathcal{O}$, operations between classes are performed so as to respect types: two objects can belong to a same class if they have the same type.

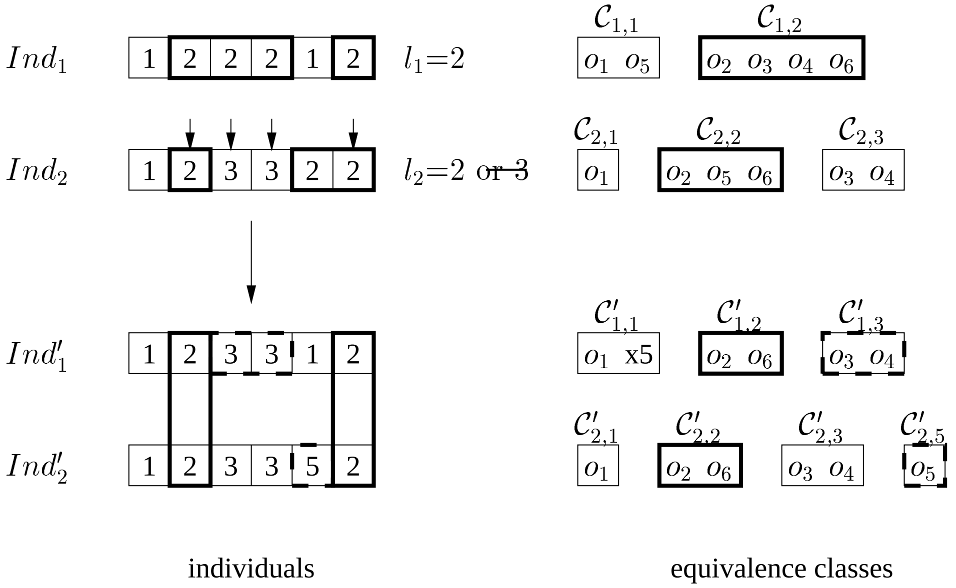

Figure 7: Intersection

3 Application to Propositionalization

As stated in Sect. 1, the problem of studying symbolic constraints between variables in propositionalization can be solved in such a way. In that case, the objects of $\mathcal{O}$ are variable symbols occurring in the pattern, and learning the equality links between variables is solved by learning a partition between variables.

3.1 Representing a Clause

We call a *pattern* the list of relations given, as well as the corresponding general clause.

We consider patterns that are non recursive definite program clauses, without any function symbols, in which all occurrences of variable are different, except if some equality relations between some of these occurrences are fixed at the beginning. To simplify, we will suppose that no equality is fixed, so that an occurrence of variable of the pattern will correspond to a symbol variable of this pattern.

We are going to extend the encoding used to learn a partition of a set of objects, to learn a partition of the set of variable symbols of a pattern. An individual encodes a clause as described below.

We will call *simplified clause*, a clause based on the schema, from which all relations that do not contain any variable linked to the head have been extracted.

A gene corresponds to a function symbol of the pattern, and a class to a symbol of variable appearing in the non simplified clause.

Let us denote by $\mathcal{S} = \{X_1, \ldots, X_n\}$ the set of symbols of variable belonging to the pattern. An individual is an array Ind indexed from 1 to n, where n is the number of occurrences of distinct variables of the pattern. For all $i \in \{1, \ldots, n\}$, $Ind[i]$ is associated to the variable X_i and its value is the label of its class. A class label is given by the smallest index among the variables of the pattern it contains.

3.2 Decoding an Individual

An individual Ind defines a substitution θ on variables of the pattern as follows.

Each class $\mathcal{C}_i$ (class labelled i) of the partition coded by the individual corresponds to a substitution ρ of a set of variables of the pattern by a variable symbol. The set of substituted variables is the set $\{X_j \mid j \in \{1,\ldots,n\},\ Ind[j] = i \text{ and } j \neq i\}$ and the variable which is substituted is X_i. Thus $\rho = \{X_j/X_i \mid j \in \{1,\ldots,n\},\ Ind[j] = i \text{ and } j \neq i\}$.

We can then define θ as the union of the substitutions corresponding to the classes of the partition coded by Ind. Let us recall that there exists a class $\mathcal{C}_i$ if $Ind[i] = i$. Thus

$$\theta = \bigcup_{i \in \{1,\ldots,n\},\ Ind[i]=i} \{X_j/X_i \mid j \in \{1,\ldots,n\},\ Ind[j] = i \text{ and } j \neq i\}$$

.

Let us note that an individual defines a unique non simplified clause, and a non simplified clause can only be coded by a unique individual.

However, an individual defines a unique simplified clause, but a simplified clause may be coded by several individuals if the simplified clause is different from the non simplified one.

3.3 Operators Effects

Let us now study the effects of such operators in terms of generalization or specialization of the rules represented by the individuals individuals.

- Isolating a variable: if the variable belongs to a class, then isolating it cuts equality relations between it and the other variables of the class, so that the rule coded by the parent is generalized.
- Moving a variable across classes: if the variable is alone in a class, then adding it to another class creates new equality relations, so that the new rule is more specific; if it is not alone, some equalities are destroyed, and new ones are created and the rule will be neither refined, nor generalized. A new point of the search space is thus explored.
- Splitting a class into two ones: in this case, all equality relations are destroyed between variables separated by the split; this generalizes the parent rule.
- Merging two classes: equality relations are created between all variables of the two classes and the rule coded by the new individual is more specific.
- Union: this operator adds equality relationships between variables of the two classes gathered, so it realizes specialization.
- Intersection: if the classes involved do not contain a unique variable, variables that are not common to them are separated from the other in their respective individuals, so that it cancels existing relations and generalizes the rule.

Our operators thus realize generalization or specialization on individuals to produce new rules, guided by the information contained in these individuals, or explore new parts of the search space. Isolating and moving a variable do that in a random manner. Merging two classes and splitting one exploit the information contained in an individual, to generate a new partition with possibly better equivalence classes. Finally union and intersection merge the information contained in two individuals concerning one class in each individual, to create two new partitions that can gather good equality relationships from the previous ones and thus be better.

3.4 Preliminary Experiments

We have implemented this approach in C++, and used the Database Management System PostgreSQL. Our prototype has been tested on the example given in Sect. 1, consisting in learning the concept $grand\_father$. Our database is composed of four tables: one for the relation $father$, one for the relation $mother$, one for storing the examples and one for storing the counter-examples. The probabilities used for the operators in our experiments are the following:

- union: 0.35,
- intersection: 0.35,
- isolating a variable: 0.1,
- adding a variable: 0.1,
- merging two classes: 0.05,
- splitting a class: 0.05.

For example, it has been tested providing it four relations: $grand\_father$, $mother$, $father$ and $mother$, with a population of 20 individuals.

For the time being, we are only searching for a conjunctive description of the concept, that covers at least one example and rejects many couter-examples. Our fitness function thus gives a null quality when the individual does not cover any example, and if any is covered, gives more weight to the rejection of counter-examples than to the coverage of examples, so that we favour consistent solutions.

Consistent definitions for $grand\_father$ emerge rapidly, as for instance the solution 0 1 2 1 0 2 6 7 which corresponds to one of the possible definitions of a maternal grandfather.

More experiments are on progress, particularly for studying the parameters of the system.

4 Related Works

As already written in Sect. 1, our work can be compared to the approach presented in [12], since both works address the same task of searching the subsumption lattice by means of GAs. Nevertheless, we propose a more compact representation of the individuals: the links between N variables are represented in our approach by an individual of length N, whereas it is represented in [12] by the top triangle of a matrix. The operators we propose are based on set operations whereas in their framework they are based on binary matrixes operations.

Many works have dealt with applying GA to Concept Learning in an attribute-value representation. Some attempts have been made in first order representations. Nevertheless, the problem is much more complicated than with attribute-value representations. Indeed, when learning in an attribute-value context, the pattern on which a rule is based is known and can be represented by a fixed-length individual. When the description of the concept has to be expressed in first order logic, we cannot determine in advance the form of the optimal rule (which literal occurs in, which variables of these literals are equal, . . .). In some genetic relational concept learners, a model for the rule is provided to the system: it is given by the user (REGAL [5], G-NET [6]) or based on a seeding example (SIAO1 [2]). In the first case, a pattern (called a template) is a conjunction of literals in which an argument is replaced by a domain. It fixes the links between the variables in the literals, and the system learns only the values allowed among those of the given domains. For example, the template $P(X, Y, [v_{p_1}, \ldots, v_{p_m}]) \wedge Q(Y, [v_{q_1}, \ldots, v_{q_n}]))$ indicates that the second variable of P and the first variable of Q are linked, that the third argument of P can take the values $v_{p_1}, \ldots, v_{p_m}$ and the second argument of Q the values $v_{q_1}, \ldots, v_{q_n}$. In SIAO1, relations between variables are modified by mutations when changing a bit to a variable (and replacing with probability the other occurrences of the past symbol by the new one in the individual), or when applying one-point crossover (but its effect on the exploitation of relations between variables seems more limited) or a generalization operator. For example, the seeding example $P(a, b) \wedge Q(b)$ could be generalized by mutation into $P(a, Y) \wedge Q(Y)$ or $P(a, Y) \wedge Q(b)$. In this approach relations between variables evolve, but the operators realize only generalization.

Let us notice that by the use of a pattern, such works can be seen as based on propositionalization.

5 Conclusion

In this paper, we have mainly focused on the subtask of learning symbolic constraints between variables in a pattern by means of a genetic algorithm. Such a work could then be embedded in a propositionalization framework. A possible way of achieving this is for instance to use a star approach [9], as follows:

1. choose a seed example e
2. define a starting pattern P that covers e
3. refine P by learning symbolic links
4. if some examples are not covered, then iterate at Step 1

Propositionalization is an interesting solution to the complexity problems linked to the size of the search space. Nevertheless, once a pattern is given, the database of positive and negative instantiations of this pattern must be built which in some degree leads to complexity problems comparable to that of Data Mining systems flattening a relational database into a single one. We have previously studied the interest of parallelizing GAs [4] and we have shown that under some conditions it was interesting to distribute the database on several processors. Other approaches have been proposed to deal with this problem as for instance stochastic sampling [11].

The genetic approach that we present is original in the sense that instead of representing in an individual whether two variables are equal or not, we encode the equivalence classes between variables and we define genetic operators that manipulates equivalence classes.

Premiliminary results are encouraging. Ongoing works aim at validating our approach on more complex problems and at finding an optimal way to control the application of our operators.

References

[1] E. Alphonse. Propositionnalisation slective en apprentissage relationnel. In *Confrence d'Apprentissage CAp'2000*, Saint-Etienne, June 2000.

[2] S. Augier, G. Venturini, and Kodratoff Y. Learning first order logic rules with a genetic algorithm. In *Proceedings of KDD'95*, pages 21–26. AAAI Press, 1995.

[3] J. N. Bhuyan, V. V. Raghavan, and V. K. Elayavalli. Genetic algorithm for clustering with ordered representation. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[4] A. Braud and C. Vrain. A study of the scalability of a data mining genetic algorithm on a distributed database. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 753, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

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

[6] A. Giordana, L. Saitta, and G. Lo Bello. A coevolutionary approach to concept learning. In *Proceedings of ISMIS-97*, volume 1325 of *LNAI*, pages 257–266. Springer, 1997.

[7] D. A. Jones and M. A. Beltramo. Solving partitioning problems with genetic algorithms. In L. B. Belew, R. K.; Booker, editor, *Proceedings of the ICGA-91*, pages 442–449, San Diego, CA, July 1991. Morgan Kaufmann.

[8] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming: Techniques and Applications.* Artificial Intelligence. Ellis Horwood (Simon & Schuster), 1994.

[9] J. G. Michalski, R. S., Carbonell and T. M. (Eds.). Mitchell. *Machine Learning, An Artificial Intelligence Approach, Volume II.* Morgan Kaufmann, Los Altos, CA, 1986.

[10] C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press, 1996.

[11] M. Sebag and C. Rouveirol. Tractable induction and classification in first order logic via stochastic matching. In *Proceedings of IJCAI-97*, pages 888–893, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.

[12] A. Tamaddoni-Nezhad and S. Muggleton. Searching the subsumption lattice by a genetic algorithm. In *Proceedings of ILP-2000*, volume 1866 of *LNAI*, pages 243–252. Springer-Verlag, 2000.

[13] J.-D. Zucker and J.-G. Ganascia. Learning structurally indeterminate clauses. In David Page, editor, *Proceedings of ILP-98*, volume 1446 of *LNAI*, pages 235–244, Berlin, July 22–24 1998. Springer.