A Relational Data Mining Tool Based on Genetic Programming

Lionel MARTIN, Frédéric MOAL, Christel VRAIN

LIFO, Université d'Orléans,
rue Léonard de Vinci, BP 6759, 45067 Orleans cedex 02, France
{martin,moal,cv}@lifo.univ-orleans.fr

Abstract. In this paper, we present a Data Mining tool based on Genetic Programming which enables to analyze complex databases, involving several relation schemes. In our approach, trees represent expressions of relational algebra and they are evaluated according to the way they discriminate positive and negative examples of the target concept. Nevertheless, relational algebra expressions are strongly typed and classical genetic operators, such as mutation and crossover, have been modified to prevent from building illegal expressions. The Genetic Programming approach that we have developed has been modeled in the framework of constraints.

1 Introduction

In this paper, we propose a Data Mining tool based on Genetic Programming. At present, most systems rely on the universal relation assumption which hypothesizes that a database is composed of a unique relation scheme [FPSSU96]. This is not realistic for real applications, since this leads to a huge relation which is hardly tractable. The Data Mining task we address in this paper is a discrimination task that can be stated as follows.

Given:
- a database scheme $\mathcal{R}$ over a set U of attributes
$\mathcal{R} = \{r_1(a_{11}:d_{11}, \ldots, a_{1k_1}:d_{1k_1}), \ldots, r_p(a_{p1}:d_{p1}, \ldots, a_{pk_p}:d_{pk_p})\}$, where r_i is a relation of arity k_i, a_{ij} is the jth attribute of the relation r_i and d_{ij} is the domain of the attribute a_{ij},
- a concept to learn defined by:
  - a positive example relation e^+ defined by the schema :
    $e^+(a_1:d_1, a_2:d_2, \ldots a_n:d_n)$
  - a negative example relation e^- defined by the same schema as e^+,

find a SQL request which recognizes the positive examples and rejects the negative ones.

Table 1: The Discrimination Task

As has been stated in the task, we aim at analyzing complex databases, involving several relation schemes and therefore, we relax the classical "universal relation scheme assumption"[FPSSU96]. To achieve this and considering

the size of the search space, we have chosen a Genetic Programming approach [Koz92, Koz94, SJB+93], that enables to stochastically explore the search space. Genetic-based approaches have already been proposed for Data Mining but, most of them use genetic *algorithms* encoding attribute-value representations [GSB97] and consequently, relying on the universal relation assumption. Some approaches [AVK95] have extended genetic algorithms to first order representations, but examples are represented by conjunctions of atoms before execution, which allows to code them with strings of fixed length. In our case, examples are only given by a relation and we must find the relations that are involved in the definition of the given concept. To our knowledge, the only Data Mining approach based on Genetic Programming is described in [RE96], but it is applied to object-oriented databases.

Let us first shortly recall the basic principles of Genetic Programming. It is an optimization method used for finding nearly optimal solution(s) to a problem, given a *fitness function*. The hypotheses are coded by trees and are called individuals; initially, a population of individuals is randomly built and then new populations are iteratively generated from the previous one by applying one of the 3 following operations:

- **reproduction:** an individual of the generation i is added to the generation $i+1$,
- **mutation:** before an individual of the generation i is added to the generation $i+1$, one of its sub-trees is replaced by a randomly generated sub-tree,
- **crossover:** given 2 individuals belonging to the generation i, 2 new individuals are added to the generation $i+1$ obtained by exchanging one sub-tree of each initial individual.

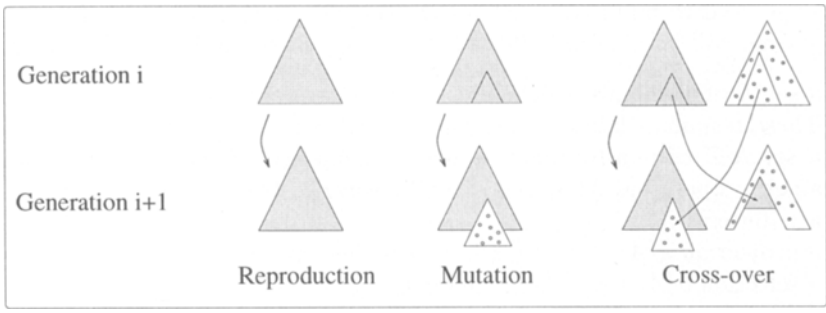

Fig. 1. Genetic operations ($\triangle$ represent trees and sub-trees)

The number of individuals in a population, the number of generations and the rate of each operation performed during a generation are fixed by parameters before learning. Moreover, for each genetic operation, individuals are chosen according to a probability proportional to their fitness.

The solution of the problem is given by the best individual that has been built. Let us notice that this result depends heavily on the values of the given parameters and on the generation of the initial population.

For our purpose, we have chosen to represent individuals of the genetic program by relational algebra expressions which enable to get results that can be both easily understood by an expert and easily translated into SQL requests for evaluation. Nevertheless, this leads to an important drawback from the point of view of Genetic Programming: relational algebra expressions are strongly typed, each node of a tree represents a relation with a fixed arity and predefined domains, therefore crossover or mutation operators can lead to syntactically or semantically incorrect expressions. Let us consider for instance, a tree representing a projection on the attribute *age* of the relation *customer* : a mutation which changes the relation *customer* into the relation *purchase* leads to an incorrect tree.

Typed Genetic Programming as defined by [Koz94] or by [Mon95] could not be used here, since a node of a tree represents a relation and its type is a list of domains: the probability to randomly select two trees with the same type is very low.

We have developed a new Genetic Programming approach and formalized it in the framework of constraints.

The paper is organized as follows. In Section 2, we present how the Data Mining task, given in Table 1, has been formalized in a Genetic Programming framework. Section 3 presents some experiments. Section 4 concludes and discusses futher works.

2 A constraint based model

2.1 Individual Representation

A tree (also called an individual) is the basic structure of our approach. The language used to build trees defines the search space, i.e. the set of hypotheses the system will be able to build. This language is syntactically defined by:

- a set $\mathcal{T}$ of terminals which is included in the database relations set, $\mathcal{T} \subset \mathcal{R}$. These terminals label the leaves of the trees ;
- a set $\mathcal{NT}$ of non-terminals which is composed of the following relational algebra operators : (A_i represents a *variable* that will refer to underlying attributes)
  - projection $\pi[A_1, \ldots, A_n]$, where n is the arity of e^+,
  - selection $\sigma[A_1\ Op\ A_2]$, where $Op \in \{=, >, <\}$,
  - selection $\sigma[A_1\ Op\ V]$, where $Op \in \{=, >, <\}$ and V is a value occurring in the database,
  - join $\bowtie [A_1, A_2]$,
  - product $\times$.

  $\bowtie [A_1, A_2]$ and $\times$ are the only operators with arity 2 ; such nodes require two subtrees.The A_i are *variables* that will refer to underlying attributes.

Let us notice that a same result can be expressed by different trees, e.g. $\bowtie [A_1, A_2](r_1, r_2)$ and $\sigma[A_1 = A_2](r_1 \times r_2)$. Such trees are not equivalent since applying mutation and crossover operator does not lead to the same individuals.

Each tree and each subtree represent a relation. For sake of simplicity, we refer to an attribute in a relation by an integer representing the position of that attribute in the underlying relation. It is called the attribute name or the attribute reference.

To deal with the problem of semantically incorrect trees, we add a set of constraints to each node of the tree, constraints that must be satisfied to get semantically correct expressions. In a first step, the expressions A_i appearing in a tree represent *variables* that must be instantiated by attribute references in a second step. A tree will be correct, if the instantiation of the variables A_i satisfies the constraints associated to each node.

The set of constraints associated to a node is defined as follows: (*arity* and *dom* are functions which respectively give the arity of a relation and the domain of a given attribute reference k)

A node	*Its set of constraints*
$\pi_{[A_1,\ldots,A_n]}(S)$	$\{1 \leq A_i \leq arity(S) \mid \forall i \in [1..n]\}$
$\sigma_{[A_1 Op V]}(S)$	$\{1 \leq A_1 \leq arity(S), dom(A_1, S) = dom(V)\}$
$\sigma_{[A_1 Op A_2]}(S)$	$\{1 \leq A_1 \leq arity(S),$ $1 \leq A_2 \leq arity(S), dom(A_1, S) = dom(A_2, S)\}$
$\bowtie_{[A_1,A_2]}(S_1, S_2)$	$\{1 \leq A_1 \leq arity(S_1), 1 \leq A_2 \leq arity(S_2),$ $dom(A_1, S_1) = dom(A_2, S_2)\}$
$\times(S_1, S_2)$	$\emptyset$
r_i	$\emptyset$

An individual is therefore a tree over $\mathcal{T} \cup \mathcal{NT}$ and a valuation ϑ which associates an attribute reference to each variable A_i and which satisfies the constraints over each of its node. This ensures that the relational algebra expression, represented by the tree is correct. In the following and without loss of generality, we suppose that each variable appearing in a tree t has been renamed in order to be unique. Consequently, satisfying the constraints of each node of t is equivalent to satisfying the set of all constraints of t. From a logical point of view, ϑ is one of the solutions to the set of constraints of t.

2.2 Individual generation

The individual generation algorithm is a recursive function. A node is randomly generated and subtrees are generated for this node, until a tree and a satisfiable set of constraints is completely built. We use a constraint solving system which is supposed to be correct, i.e., not to reject satisfiable set of constraints. The major advantage of using a constraint solving system is that the generation immediately stops when a partial tree leads to a deadlock. Once a tree has been completely constructed, the constraint solving system is called to get a set of correct instantiations of the variables of the tree. We then randomly choose *one* valuation, given a set ϑ of instanciations *variable* = *value*.

The result of the generation process is a pair (t, ϑ), where t is a tree over $\mathcal{T} \cup \mathcal{NT}$ which contains variables denoted by $A_1, \ldots, A_k$, and where ϑ is a set

$\{A_1 = ref_1, \ldots, A_k = ref_k\}$, where ref_i are references to attributes of relations that appears in t.

Let us notice that the system has some initial constraints, namely the domains of the target concept given by the relation e^+. Such constraints must always be satisfied and cannot be relaxed.

2.3 Crossover and mutation : a constraint solving problem

Let us consider an individual (t, ϑ) and a subtree s of t. The main problem when dealing with mutation and crossover operators occurs when replacing the subtree s by another individual defined by the tree s' and the valuation $\vartheta_{s'}$ (which comes from another individual in case of crossover, or has been randomly generated in case of mutation) given the tree t', since the new set of constraints thus obtained must be still satisfiable.

Notations:

- ϑ_s denotes the restriction of ϑ to variables of s,
- ϑ_o denotes the restriction of ϑ to the rest of the tree $(t - s)$,
- ϑ_o is decomposed into two sets: ϑ_{o1} is restricted to the variables that do not refer to attributes occurring in the subtree s and ϑ_{o2} is restricted to variables which refer to attributes occurring in s (cf figure 2).

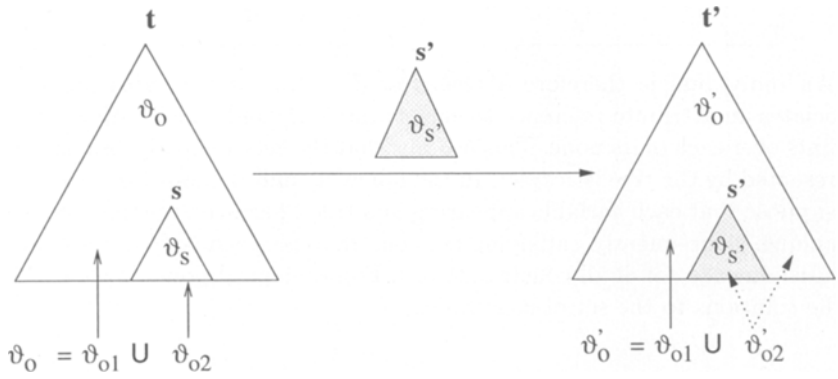

Fig. 2. Crossover and mutation: subtrees and valuations exchange

There are three possible cases :

- Case 1: there exists a valuation ϑ', which is consistent with the set of constraints defined in t' and which extends $\vartheta_{o1} \cup \vartheta_{s'}$; in other words, ϑ' gives only new values to the variables occurring in ϑ_{o2}. The result is therefore the individual (t', ϑ').
- Case 2: there exists a valuation ϑ', which is consistent with the set of constraints defined in t' and which extends $\vartheta_{s'}$ (but which does not extend ϑ_{o1}). It is no longer possible to, even partially, keep the previous valuation ϑ_{o1}. The result is therefore the individual (t', ϑ').

– Case 3: the new tree t' has no solution, even when relaxing all the valuations. In that case, the operator applied to this individual with the subtree s' fails.

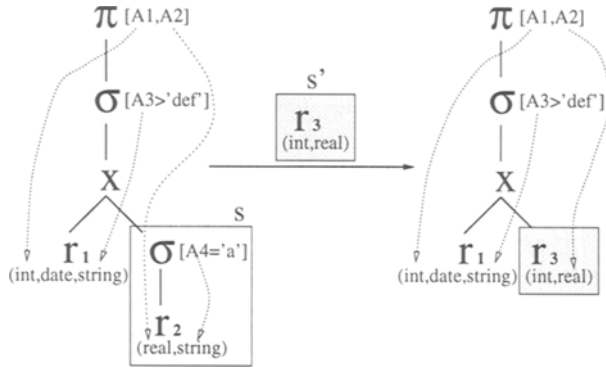

Fig. 3. ϑ_{o1} $(A_1 = 1, A_3 = 3)$ remains valid (Case 1)

For instance, in Figure 3, the resulting valuation is constructed with ϑ_{o1} which assigns to A_1 the first attribute of r_1 and to A_3 the third attribute of r_1 and with a new valuation for A_2 to the second attribute of r_3 (the fact that A_4 is assigned the second attribute of r_2 is no longer useful).

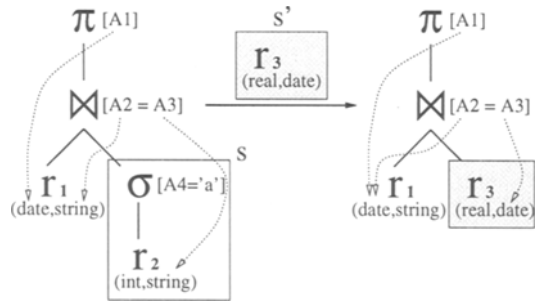

Fig. 4. ϑ_{o1} $(A_1 = 1, A_2 = 2)$ must be changed. (Case 2)

Figure 4 illustrates the second case: the initial valuation ϑ_{o1} is no longer consistent, because A_2 refers to a string attribute, and there is no string in the new tree s'. So, keeping this valuation makes the constraint $(dom(A_2) = dom(A_3))$ on the join node not satisfiable. To solve this problem, we relax the valuation ϑ_{o1}, and the new set of constraints becomes satisfiable with $dom(A_2) = dom(A_3) = date$.

The last case is similar to the previous one, but even after relaxing ϑ_{o1}, the constraint set remains unsatisfiable.

In our application, a less random version of the mutation operator has also been implemented: it allows to generalize or specialize an individual.

3 Preliminary results

This approach has been implemented in C. The evaluation of individuals is performed through a standard interface. According to the choice of the user, either Oracle is called (by means of PRO*C) and requests are executed, or a module implemented in SWI-PROLOG for data represented by ground atoms is used. The fitness function has been defined so that it penalizes individuals that cover negative examples. The usual genetic programming parameters are the Reproduction rate (here set to 20%), the Crossover rate (60%) and the Mutation rate (20%).

3.1 Michalski's 10 trains

This example [MMPS94] is a classical Machine Learning example. It has only a few instances, but it allows to test the convergence of the algorithm in a relational case : there are 8 relations (*car, load, in-front, ccont, ...*) with redundancies.

The dataset is composed of 10 trains, and the goal is to discriminate between 5 trains (positive examples) going in the east direction and 5 trains (negative examples) going in the west direction. The population contains 1000 individuals and selection is restricted to the equality operator. The algorithm converges to the following *conjunctive* definition for east trains :

```
SELECT r0.idtrain
FROM ccont r0, infront r1, load r2, load r3
WHERE (r0.idwagon=r1.idwagon) AND (r2.idwagon=r1.idwagon)
  AND (r3.idwagon=r1.idwagon2) AND (r2.load_shape='triangle');
```

It is important to notice that this definition is a complete and consistent solution and that it does involve several relations (and several occurrences of the same one).

3.2 The mushrooms

This database [Sch87] has only *one* relation, called *mush*, which contains all data about 8124 mushrooms. The *mush* relation has 22 attributes with discrete values. The goal is to discriminate the edible mushrooms (4208) from the poisonous one (3916).

Here the set of non-terminal is restricted to projection and constant selection (since there is a single relation). The trees are linear, crossover and mutation operators always succeed. Such an experiment enables to test the performance of the system in an attribute/value case, with much more instances. Two conjunctive solutions emerge, depending on the relative weight given to the number of negative covered examples in the fitness function : $3408e^+/120e^-$ covered by the first one (80%/3%), and $2496e^+/0e^-$ by the second one (59%/0%).

3.3 The 100 trains example

This example [MMPS94] is similar to the 10 trains example, but there are 100 unclassified trains, and about 300 cars. The dataset is *a priori* unclassified. We have invented a concept and then tested how our system could learn the right definition of this concept.

We have chosen the following partially defined concept:
– the positive examples $e+$ are defined by the relation "SELECT r1.idtrain FROM car r1 WHERE (r1.position = 1) AND (r1.wheels = 3)" (16 trains),
– the negative examples $e-$ are defined by "SELECT r1.idtrain FROM car r1, car r2 WHERE (r1.wheels $\neq$ 3) AND (r1.idtrain = r2.idtrain) AND (r2.shape = 'rectangle') AND (r2.roof_shape = 'flat')" (38 trains).

Here, the population contains 250 individuals. Since the target concept is a "simple" conjunctive one, the best solution is reached after five generations in average.

4 Conclusion

The experiments show that the system that we have developed is able to handle several relations, even on quite important collections of data. Further experiments must still be achieved to test the efficiency on very large collection of data and to improve the quality of the fitness function.

The method proposed in this paper brings two kinds of contribution:
– a formalization of the search space in terms of relational algebra,
– an exploration of such a search space by a Genetic Programming approach.

The first point has already been studied in [BR97]. Nevertheless, the two approaches differ: in [BR97], a normal form of relational algebra expression is used and the algorithm relies on an exhaustive evaluation of all possible products of relations in the database.

The approach that we have chosen associates to each node of a tree a set of *internal* constraints, which defines the space of all correct individuals. When dealing with a particular problem, the user usually has knowledge about the solution, or knowledge about what is not a solution. This is usually coded by biases. A first method to express biases is to restrict the set $\mathcal{T} \cup \mathcal{NT}$, but this can be too drastic and this does not allow to express fine knowledge. Another method would be to extend the set of constraints with *external* constraints, defined by the user and associated to the root of each tree. For instance, limiting the height of the trees would be simply written by adding the constraint $height(t) < 7$ at the root of a tree t (where *height* is a function recursively defined over the nodes). This approach has two major interests:

– extensibility: this would allow the user to define new biases over the initial search space,

– simplicity: this would provide us with an unified way to express the search space (constraint trees) and biases over it.

Neverthess, we must now study the class of biases that can be represented by constraints.

Another important point is learning disjunctive concepts. For the time being, we have focused on conjunctive concepts, and a parallel architecture, splitting the whole population into subpopulations (islands), was mainly used to promote species formation, i.e. the emergence of different solutions. We would like to use such an architecture for learning disjunctive concepts, as done in the system Gnet [GSB97]. The final solution will be the union of *some* of the best conjunctive ones.

References

[AVK95] S. Augier, G. Venturini, and Y. Kodratoff. Learning first order logic rules with a genetic algorithm. In *Proceedings of the First International Conference on Knowledge Discovery & Data Mining (KDD'95)*, pages 21–26, Canada, August 1995.

[BR97] Hendrik Blockeel and Luc De Raedt. Relational knowledge discovery in databases. In Stephen Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming (ILP-96)*, volume 1314 of *LNAI*, pages 199–211, Berlin, August 26–28 1997. Springer.

[FPSSU96] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* MIT Press, Mento Park, 1996.

[GSB97] A. Giordana, L. Saitta, and G. Lo Bello. A coevolutionary approach to concept learning. In Zbigniew W. Raś and Andrzej Skowron, editors, *Proceedings of the 10th International Symposium on Foundations of Intelligent Systems (ISMIS-97)*, volume 1325 of *LNAI*, pages 257–266, Berlin, October 15–18 1997. Springer.

[Koz92] J. R. Koza. *Genetic Programming On the programming of computers by means of natural selection.* MIT Press, Cambridge, Massachusetts, 1992.

[Koz94] J. R. Koza. *Genetic Programming II Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, Massachusetts, 1994.

[MMPS94] D. Michie, S. Muggleton, D. Page, and A. Srinivasan. To the international computing community: A new East-West challenge. Technical report, Oxford University Computing laboratory, Oxford,UK, 1994.

[Mon95] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[RE96] Tae-Wan Ryu and Christoph F. Eick. Deriving queries from examples using genetic programming. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *The Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 303–306, Portland, Oregon, USA, August 2-4 1996. AAAI.

[Sch87] J. C. Schlimmer. Concept acquisition through representational adjustment. Technical Report ICS-TR-87-19, University of California, Irvine, Department of Information and Computer Science, July 1987.

[SJB+93] W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In Pavel B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667 of *LNAI*, pages 442–459, Vienna, Austria, April 1993. Springer Verlag.