Mathware & Soft Computing 7 (2000) 337-350

A Niching Scheme for Steady State GA-P and its Application to Fuzzy Rule Based Classifiers Induction

L. Sánchez Ramos and J.A. Corrales González
Universidad de Oviedo. Departamento de Informática.
Campus de Viesques, 33207 Gijón. Asturias
luciano,ja@lsi.uniovi.es

Abstract

A new method for applying grammar based Genetic Programming to learn fuzzy rule based classifiers from examples is proposed. It will produce linguistically understandable, rule based definitions in which not all features are present in the antecedents. A feature selection is implicit in the algorithm.

Since both surface and deep structure will be learned, standard grammar based GP is not applicable to this problem. We have adapted GA-P algorithms, a method formerly defined as an hybrid between GA and GP, that is able to perform a more effective search in the parameters space than canonical GP do. Our version of GA-P supports a grammatical description of the genotype, a syntax tree based codification (which is more efficient than parse tree based representations) and a niching scheme which improves the convergence properties of this algorithm when applied to this problem.

Keywords: Genetic Programming, Genetic Fuzzy Systems, Fuzzy Classifiers.

1 Introduction

Genetically inducting a fuzzy rule base classifier consists in evolving the definition of the fuzzy partitions and the surface structure [2]. Linear genotypes use a rigid codification in which the antecedent of all rules is an "and" combination of asserts that relate every feature to a linguistic label. Non linear genotypes, mostly tree shaped ones —genetic algorithms with this representation are called *genetic programming* (GP)— admit a more general structure. It was recently shown that GP can be regarded as a particular case of GA with variable length representation, because there always can be found a context free grammar in which every tree shaped individual corresponds to one valid chain [5]. The version of genetic algorithms that explicitly deal with chains in a grammar is called *grammar based*

GP. Many different structures of fuzzy classifiers can be described with grammars, hence they can be induced with grammar based GP.

Genotype in grammar based GP is the parse tree of the chain, thus genetic operators act on parse trees. This is rather inefficient in time (chain needs to be compiled before its fitness is evaluated) and in space (a chain is contained in the frontier of its parse tree). This overload is significant when searching for moderately long chains, as is the case in fuzzy classifiers induction. We propose to use a labeled syntax tree that allows to interpret the chains without the need of compiling them, resulting in an structure similar to that of strongly typed GP [9]; production rule names will act as data types.

Fuzzy classifiers depend on a linguistic description (the surface structure) and fuzzy partitions of features (the deep structure) [12]. Fuzzy partitions are defined by real numbers. Since grammar based GP is not efficient finding accurate numerical expressions, it may be helped with other numerical optimization methods. We will combine it with fixed length, real coded GA in a GA-P algorithm. GA-P is a graph-based representation (it was formerly defined as an hybrid between GA and GP [6]) which is more efficient than GP in this task. It will be shown how to apply GA-P algorithms to fuzzy classifier induction. It is remarked that GP is regarded as a particular case of GA in this paper, thus GA-P algorithms will in turn be described as a set of genetic operators that extend GA to certain non linear genotype. It is also proposed a niching method that improves the convergence rate of the basic GA-P algorithm.

This paper is organized as follows: first, we will give one precise meaning to the linguistic representation of a fuzzy rule (section 2). Then we will state the set of GA operators that give rise to the GA-P algorithm (section 3), along with a niching scheme efficient for inducting fuzzy rule bases (section 4). By last, numerical results are shown in which behaviors of GA-P with/without niches and GA-P vs. other algorithms are compared.

2 Semantic of a fuzzy rule base

The semantic of a fuzzy rule base will be defined, in terms of a fuzzy relation, by recursively applying the conditions that follow. Recall that a fuzzy classifier is defined by a fuzzy relation that assigns values between 0 and 1 to tuples of $m+1$ values. Each tuple reflects the confidence we have on a combination on linguistic values of all features corresponds to one of the classes; in other words: the first value is a linguistic label defined over the first feature, the second value is a label in the second feature, and so on, until we get the value number $m+1$, which is the class number.

2.1 Single and degenerate fuzzy rules

Let $\widetilde{l}_{ij}$ be the linguistic label number "j" in the feature number "i". The rule

$$\text{“}\texttt{if } x_i \texttt{ is } \widetilde{l}_{ij} \texttt{ then class is } c_k \texttt{ with conf. } \alpha\text{”}$$

defines the relation that follows:

$$R(x_1, \ldots, x_m, c) = \begin{cases} \alpha & \text{if } x_i = \widetilde{l}_{ij} \text{ and } c = c_k \\ 0 & \text{otherwise.} \end{cases}$$

The degenerate rule

“`if` α `then class is` c_k”

defines the relation

$$R(x_1, \ldots, x_m, c) = \begin{cases} \alpha & \text{if } c = c_k \\ 0 & \text{otherwise} \end{cases}$$

2.2 Semantics of logical expressions in antecedents

If the rules

“`if` A `then class is` c_k”

“`if` B `then class is` c_k”

define the relations R_A and R_B, then the rules

`if` $A \wedge B$ `then class is` c_k

`if` $A \vee B$ `then class is` c_k

define the relations

$$R_{A \wedge B}(x) = \min(R_A(x), R_B(x))$$

$$R_{A \vee B}(x) = \max(R_A(x), R_B(x))$$

respectively. As a consequence of this, the rule

“`if` $A \wedge \alpha$ `then class is` c_k”

describes the same relation as the rule

“`if` A `then class is` c_k `with confidence` α”.

Therefore, if we allow the presence of logical constants in the antecedents, all relations can be described by rules with confidence 1. For this reason, we do not use degrees of confidence in the linguistic expression of the rules.

2.3 Semantics of a concatenation of rules

The concatenation of N rules defined by relations $R_1, R_2, \ldots, R_N$ define the relation $R(x) = \max\{R_1(x), \ldots, R_N(x)\}$. Two rules with the same consequent can be combined in one. The construction

`if` A `then class is` c_k

`if` B `then class is` c_k

is identical to

`if` $A \vee B$ `then class is` c_k

CLASSIFIER $\rightarrow$ if CONDITION then class is c_1
  if CONDITION then class is c_2
  ...
  if CONDITION then class is c_{N_c}
  LOGICAL-CONS
  PARTITION-CONS
CONDITION $\rightarrow$ ASSERT$_1$ $|$ ASSERT$_2$ $|$ ... $|$ ASSERT$_m$
  | (CONDITION $\vee$ CONDITION)
  | (CONDITION $\wedge$ CONDITION)
  | K$_1$ | K$_2$ | ... | K$_{N_c}$
ASSERT$_1$ $\rightarrow$ left-trapezium(x$_1$,K$_{11}$,K$_{12}$) |
  triangle(x$_1$,K$_{11}$,K$_{12}$,K$_{13}$) |
  ...
  right-trapezium(x$_1$,K$_{1n_1-1}$,K$_{1n_1}$)
...
ASSERT$_m$ $\rightarrow$ left-trapezium(x$_m$,K$_{m1}$,K$_{m2}$) |
  triangle(x$_m$,K$_{m1}$,K$_{m2}$,K$_{m3}$) |
  ...
  right-trapezium(x$_m$,K$_{mn_m-1}$,K$_{mn_m}$)
PARTITION-CONS $\rightarrow$ K$_{11}$ ... K$_{1n_1}$ ... K$_{m1}$ ... K$_{mn_m}$
LOGICAL-CONS $\rightarrow$ K$_1$ K$_2$ ... K$_{N_c}$

Figure 1: Grammar defining the phenotype of a fuzzy rule based classifier. The genotype will consist in a acyclic graph built from the syntactic tree of the classifier. Observe that the linguistic expression comprises so many rules as classes, plus two chains of real numbers.

3 Fuzzy classifier induction with GAs

Fuzzy rules, as defined in the previous section, have a rather rigid syntax that can be cast into a context free grammar. This allow us to genetically induce fuzzy rule bases with the help of genetic algorithms defined over the chains of this grammar. In particular, we will use GA-P algorithms to search for both deep and surface structure of the rule base in one stage.

3.1 Representation of a fuzzy rule based classifier

For our purposes, a fuzzy classifier is a valid chain in the context free grammar defined by the production rules shown in figure 1. N_c is the number of classes, $x_1, \ldots, x_m$ are the features and n_i the number of linguistic terms in feature i, $i = 1, \ldots, m$.

Observe that we allow using the logical connective "or" in antecedents and permit that some features are absent from the antecedent. `left-trapezium`, `triangle` and `right-trapezium` are trapezoidal or triangular fuzzy memberships defined by two or three parameters (see Figure 2). All "K" terminal symbols are real numbers.

Figure 2: Trapezoidal and triangular membership functions and the meaning of their arguments.

Since we allow using "or" in the antecedents, there are so many rules as classes. Observe that we admit "logical constants" in the expression of the antecedent; their meaning will be explained in the next section. There are two semantic restrictions over the values of the constants. All genetic operators must preserve them:

1. Constants $\mathrm{K}_1\ \mathrm{K}_2 \ldots \mathrm{K}_{N_c}$ take values between 0 and 1.
2. The lists $[\mathrm{K}_{11} \ldots \mathrm{K}_{1n_1}] \ldots [\mathrm{K}_{m1} \ldots \mathrm{K}_{mn_m}]$, are ordered and their values must be inside the range defined for every one of the features.

For example, the surface structure of a linguistic classifier for which $N_c = 2$, $m = 2$, $n_1 = 3$, $n_2 = 3$ and all features range from 0 to 1 is as follows:

```
if x1 is MEDIUM-1 and x2 is MEDIUM-2 then class is c1 with conf. 1
if x2 is LOW-2 then class is c1 with conf. 1
if x1 is HIGH-1 then class is c2 with conf. 0.2
```

Its deep structure is defined by its surface structure plus the definition of the fuzzy partitions of both features. For instance:

```
LOW-1(x1)=trapezium-left(x1,0.1,0.2)
MEDIUM-1(x1)=triangle(x1,0.1,0.2,0.5)
HIGH-1(x1)=trapezium-right(x1,0.2,0.5)
LOW-2(x2)=trapezium-left(x2,0.0,0.5)
MEDIUM-2(x2)=triangle(x2,0.0,0.5,0.8)
HIGH-2(x2)=trapezium-right(x2,0.5,0.8)
```

We represent the deep structure of this fuzzy classifier by means of the chain that follows:

```
if triangle(x1,k11,k12,k13) and triangle(x2,k21,k22,k23)
  or trapezium-left(x2,k21,k22) then class is c1
if trapezium-right(x1,k12,k13) and k2 then class is c2
0.7 0.2 0.9
0.1 0.2 0.5 0.0 0.5 0.8
```

Figure 3: Phenotype-Genotype representation of the chain shown in section 3.1 following Geyer Schulz's approach (upper part) and ours (lower part). In Geyer's the chain is the frontier of its parse tree and genotype crossover takes place between subtrees labeled with the same symbol in the root. In our method an acyclic graph derived from the syntactic tree of chain and GA-P crossover are used. The names of the production rule that originate each subtree are written in parentheses and act as types in strongly typed GP.

Observe that the second rule makes sense in this grammar as a consequence of the semantics defined in the preceding section, but in human language it should be read as “if trapezium-right(x1,k12,k13) then class is c2 with confidence k2”.

The genotype is an acyclic directed graph formed by the syntactic tree of the chain and an array with the values of all parameters, shown in figure 3. Not all parameters need to be used; in this example, `k1`= 0.7 and `k2`= 0.9 are not referenced by any rule. This graph must be labeled with the names of the production rules that originate every subtree, which is a particular case of strongly typed GP [9].

3.2 Crossover and mutation operators

As in strongly typed GP crossover, we only interchange subtrees that are evaluated to values of the same type. Since we have defined the type of a node as the name of the production rule which originates it, typed crossover preserves grammatical correctness (i.e., any crossover of two trees is the syntactic tree of a valid chain in the grammar). Observe that this crossover can be seen as an extension of the two-point crossover to tree shaped genotype.

There are not numbers in the terminal nodes of the trees but pointers to an array (see figure 3). GA-P algorithms merge two kinds of crossover at random: the usual subtree interchange crossover (see figure 4) and intermediate crossover [10] between the arrays of real numbers that we have mentioned.

Mutation is defined as the crossover of an individual with a randomly generated one (i.e., “headless chicken crossover” [7]).

4 A niching scheme for GA-P

We have observed that in the final stages in steady state GA-P it is frequent that all individuals in population share the same linguistic expression and differ only in their numeric parameters, thus GA-P converges to a fixed length GA. This is a consequence of the nature of the GA-P algorithm; when a new linguistic expression appears, it can not be decided whether it will be a good candidate unless we optimize their numerical parameters, but this will not be done if its initial fitness is too low to be inserted in the population. The use of niches seems to be the natural correction to this behavior. Our implementation differs from standard niching GA [3] because we consider that species are groups of individuals with the same linguistic expression and equal or different parameters. We have tried similarity measures based on edition distance to define niches that contain similar but not identical individuals, to reproduce concepts as niche radius and fitness sharing, but they did not produce good results: distances between individuals and the differences between their fitness values are not correlated, as it occurs in multimodal numerical optimization, and similar individuals can have very different fitness values.

The outline of a niching steady state GA-P algorithm is shown in figure 6, and the meaning of the constants in this pseudo-code is shown in figure 5. To maintain an adequate number of species we decided to use two functions `desired_copies` and

Figure 4: Subtree crossover: Subtrees of parents with the same types (hence originated by the same production rule) can be interchanged. The array of parameters is not altered by subtree crossover; there is a specific “chain crossover” operation that is randomly alternated with subtree crossover.

Parameter	Description
`MAX_POP`	Population size
`MAX_ITER`	Maximum number of fitness evaluations
`SIZE_T`	Tournament size
`PROB_MUT`	Percentage of mutation
`PROB_INTRA`	Percentage of intra-niche crossover
`PROB_SUBTREE`	Percentage of subtree crossover/mutation
`N_MAX`	Size of the largest niche

Figure 5: Parameters in niche GA-P algorithm

```
niche_GAP
  initialize population
  while number_of_fitness_evaluations<MAX_ITER
    t[1]=randomly selected individual
    if (rnd()<PROB_INTRA) and (size_of_niche(t[1])>=SIZE_T) then
      t[2..SIZE_T]=individuals from the same niche as t[1]
      sort t by fitness
      offspring[1..2]=chain crossover(t[1..2]))
      chain mutation of offspring, with prob PROB_MUT
    else
      t[2..SIZE_T]=individuals selected at random
      sort t by fitness
      if rnd()<PROB_SUBTREE then
        offspring[1..2]=subtree crossover(t[1..2]))
        subtree mutation of offspring, with prob PROB_MUT
      else
        offspring[1..2]=chain crossover(t[1..2]))
        chain mutation of offspring, with prob PROB_MUT
      end if
    end if
    insert_and_delete(offspring[1],offspring[2],
                      t[SIZE_T-1],t[SIZE_T-2])
end niche_GAP
```

Figure 6: Basic steady state GA-P algorithm

size_of_niche. The first one relates the index of an individual in the population (which is ordered by the fitness value) with the maximum number of individuals with the same tree part as it we want to exist; details about this function will be given later in this section. The expression size_of_niche(a) counts how many individuals have the same tree part as 'a'. By comparing them, we decide if the offspring of the crossover operation is inserted or not and also if the losers in the tournaments are removed or not from the population. This way we dynamically alter the sizes of all niches as new individuals are generated.

The key of this process is the function insert_and_delete (see figure 7), which has four arguments. It tries to insert the first two ones into the population and to delete the last two ones from it. The insertions are not made if the niche which receives the individual has reached its maximum size, and the deletions are not made if the niche from which the individual should be removed is smaller than it should be. We keep track of the number of insertions and deletions, so this operation does not alter the size of the population. When more insertions than deletions are made, the excess of individuals (the worst ones in the population are chosen) is deleted.

To decide whether a niche's size is adequate we use the function desired_copies, as we mentioned before. It estimates how many copies of an individual are needed, depending on its fitness. It makes no sense to keep a large number of copies of an individual with low fitness, and conversely there should be many copies of good individuals, to accelerate the search, hence we did not define constant sized niches. Let $n_{\max}$ be the maximum size of a niche (i.e., the maximum size we allow the niche that contains the best individual in the population to have) and let us decide that the niche size of the worst individual in population is 1. We use a linear variation of size, as given by

$$n(a) = \max\left(0, \left\lfloor \frac{(n_{\max}-1)(\texttt{MAX\_POP} - o_{\text{best}})}{\texttt{MAX\_POP} - 1} + 1 \right\rfloor\right)$$

where n is the number of copies of the individual, o_{best} is the ordinal of the best individual in the same niche as a (if a is going to be the best individual in its niche, o_{best} is the ordinal that a would have in the population if it was inserted), and P is the size of the population.

5 Numerical results

In figure 9 some different fuzzy rule learning algorithms are compared to standard GA-P and niche GA-P over Pima, Cancer and Glass datasets. Dietterich's 5x2cv statistical test [4] was used to validate the effect of niching in GA-P algorithms.

In all cases the algorithm was stopped after 5000 evaluations of the fitness function. Population size (MAX_POP) is 100, tournament size (SIZE_T) is 4, probability of mutation (PROB_MUT) is 0.01, 50% in intra-niche crossover (PROB_INTRA), 50% of subtree-chain crossover and mutation (PROB_SUBTREE) and maximum niche size (N_MAX) is 10. Niching achieved better mean performance in all tests, but this difference is not always statistically relevant. It was found that the hypotheses

```
insert_and_delete(a,b,c,d)
    excess=0
    if desired_copies(a) < size_of_niche(a) then
        insert a in population
        excess=excess+1
    end if
    if desired_copies(b) < size_of_niche(b) then
        insert b in population
        excess=excess+1
    end if
    if desired_copies(c) > size_of_niche(c) then
        delete c from population
        excess=excess-1
    end if
    if desired_copies(d) > size_of_niche(d) then
        delete d from population
        excess=excess-1
    end if
    delete the last 'excess' individuals in population
end insert
```

Figure 7: Insertion in population with adaptive niching

Figure 8: Number of species in three runs of GA-P, without niches (left) and with the adaptive niching proposed in this paper (center and right). The size of the population is 100, the maximum size of a niche is 10 (center) and 50 (right). Vertical axis shows the percentages of the population that are occupied by individuals with the same tree part.

Figure 9: Comparison between the deviation of test error when repeating 10 times the learning over PIMA, CANCER and GLASS dataset and 5x2cv experimental framework. From left to right: Grammar based GA-P without niches, niching GA-P with maximum niche size of 5, 10, 20 and 50. Population size is 100.

"niching is relevant to the algorithm" is not rejected with a confidence of 99%, 87% and 80% for Pima, Cancer and Glass, respectively.

The benchmark defined in [11] was also used to contrast GA-P algorithms when inducting fuzzy classifiers and to provide numerical data of the expected classification error. Every experiment has been repeated 30 times over three different permutations of the dataset. 75% of samples were used to design the classifier, 25% to test it by cross-validation. In Figure 10 the mean values obtained are shown for every permutation along with the standard deviations of the experiments. The same framework as in the previous experiment was used except for population size and the number of iterations. 5 parallel subpopulations of size 100 were defined (migration probability was 1%). The algorithm was stopped after evaluating 50000 times the fitness function (approximately 10000 evaluations in every subpopulation).

Data relative to linear and neural classifiers error is taken from [11]. The parameter of the k-neighbors classifier was chosen to minimize the classification error evaluated with leave-one-out. Observe that the dispersion of the results obtained when training a neural network from different starting points can be higher than the dispersion of GA-P results (figure 9).

Since a classifier that decides between M classes can always be written with M rules, counting the number of rules makes no sense here. We have measured the complexity with the number of nodes in the classifier's syntax tree instead. Last column displays the mean number of characteristics a classifier use. The results are much compact than most fuzzy rule learning algorithms, but results should not be numerically compared due to different structure of rule bases in the methods that were studied and this one. The length of the linguistical expression of the final classifier induced with GA-P is much shorted with lengths of other GA induced

Dataset	K-NN	Linear Mean Dev	MLP Mean Dev	GA-P Mean Dev	Complexity Nodes	Variables Used Total
Cancer-1	1.7	2.93 0.18	1.38 0.49	3.06 1.00	37.7	4.5/9
Cancer-2	4.0	5.00 0.61	4.77 0.94	6.30 1.99	36.5	4.4/9
Cancer-3	4.5	5.17 0.00	3.70 0.52	4.50 1.79	35	4.4/9
Thyroid-1	5.95	6.56 0.00	2.38 0.35	4.92 0.94	34.5	4.1/21
Thyroid-2	6.00	6.56 0.00	1.91 0.24	4.80 0.82	37.7	4.5/21
Thyroid-3	6.50	7.23 0.02	2.27 0.32	4.88 0.88	36.6	4.3/21
Pima-1	25.5	25.83 0.56	24.10 1.91	24.92 1.89	34.1	3.7/8
Pima-2	27.6	24.69 0.61	26.42 2.26	29.68 2.05	24.3	3/8
Pima-3	23.5	22.92 0.35	22.59 2.23	24.29 2.17	25.4	3/8
Glass-1	35.8	46.04 2.21	32.70 5.34	39.62 4.73	73.9	6.7/9
Glass-2	33.9	55.28 1.27	55.57 3.70	42.11 2.78	81	5.8/9
Glass-3	35.0	60.57 3.82	58.40 7.82	41.15 4.72	68.5	5.3/9

Figure 10: Compared results between fuzzy classifiers obtained with GA-P algorithms and statistical and neural classifiers over the benchmark defined in [11].

classifiers' lengths, often by an order of magnitude.

6 Concluding Remarks

When designing fuzzy rule based classifiers, one of the main concerns is how much predictive accuracy has to be sacrificed for the sake of intelligibility. Canonical GP has been previously used to induce surface structure of fuzzy classifiers, but inducting deep structures requires combining it with a numerical optimization method, like GA-P does. We have shown that GA-P algorithms achieve from data with good numerical behavior and obtaining compact solutions, and adapted them to use a grammatical description of a fuzzy classifier and to support a niching scheme that sustains diversity in steady state GA-P.

References

[1] Banzhaf, W. et al *Genetic Programming: An Introduction.* Morgan Kaufmann, 1998.

[2] Cordón, O., Herrera, F. "A general study on genetic fuzzy systems". In Periaux J., Winter G., Galán M. y Cuesta P. (Eds.) *Genetic Algorithms in Engineering and Computer Science*, pp. 33-57. Wiley and Sons.

[3] Deb K. and Goldberg D. "An investigation of niche and species formation in genetic function optimization". In Proc. of 3rd ICGA'89, 1989.

[4] Dietterich, G. “Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms”. Neural Computation, 10(**7**), pp 1895-1924. 1998

[5] Geyer-Schulz, A. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning.* Second edition. Physica-Verlag, 1997.

[6] Howard, L., D' Angelo, D. “The GA-P: a genetic algorithm and genetic programming hybrid”. IEEE Expert, pp 11-15. 1995.

[7] Jones, T. “Crossover, macromutation, and population-based search”. Proceedings of the Sixth International Conference on Genetic Algorithms, Pittsburgh, PA, 15.-19, pp. 73-80. 1995.

[8] Koza, J., *Genetic Programming.* MIT Press, Cambridge, MA. 1992

[9] Montana, D. “Strongly Typed Genetic Programming” Evolutionary Computation, 3, pp. 199-230, 1995.

[10] Mühlenbein, H. and Schlierkamp-Voosen, D.: “Predictive Models for the Breeder Genetic Algorithm”: I. Continuous Parameter Optimization. Evolutionary Computation, 1 (1), pp. 25-49, 1993.

[11] Prechelt, Lutz. “PROBEN1 – A set of benchmarks and benchmarking rules for neural network training algorithms”. Tech. Rep. 21/94, Fakultät für Informatik, Universität Karlsruhe, 1994.

[12] Zadeh, L. “Fuzzy Logic, Neural Networks and Soft Computing”. Communications of the ACM, **37**(3), pp. 77-84. 1994.